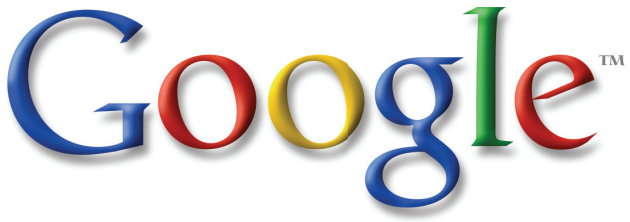


Randomized Algorithms for Data

Fast and simple methods

Alex Smola

Carnegie Mellon University, MLD

Google, Strategic Technologies

Carnegie Mellon University

Thanks

Quoc Le, Tamas Sarlos, Markus Weimer, Amr Ahmed, Alexandros Karatzoglou, John Langford, Kilian Weinberger, Anirban Dasgupta, Vishy Vishwanathan, Gideon Dror, Yehuda Koren, Javen Shi, Choon-Hui Teo, Sergiy Matusevich …

Kannelakis Prize 2012

Andrei Broder (2012)

With Moses S Charikar and Piotr Indyk, for their groundbreaking work on Locality-Sensitive Hashing that has had great impact in many fields of computer science including computer vision, databases, information retrieval, machine learning, and signal processing.

ABOUT THIS AWARD

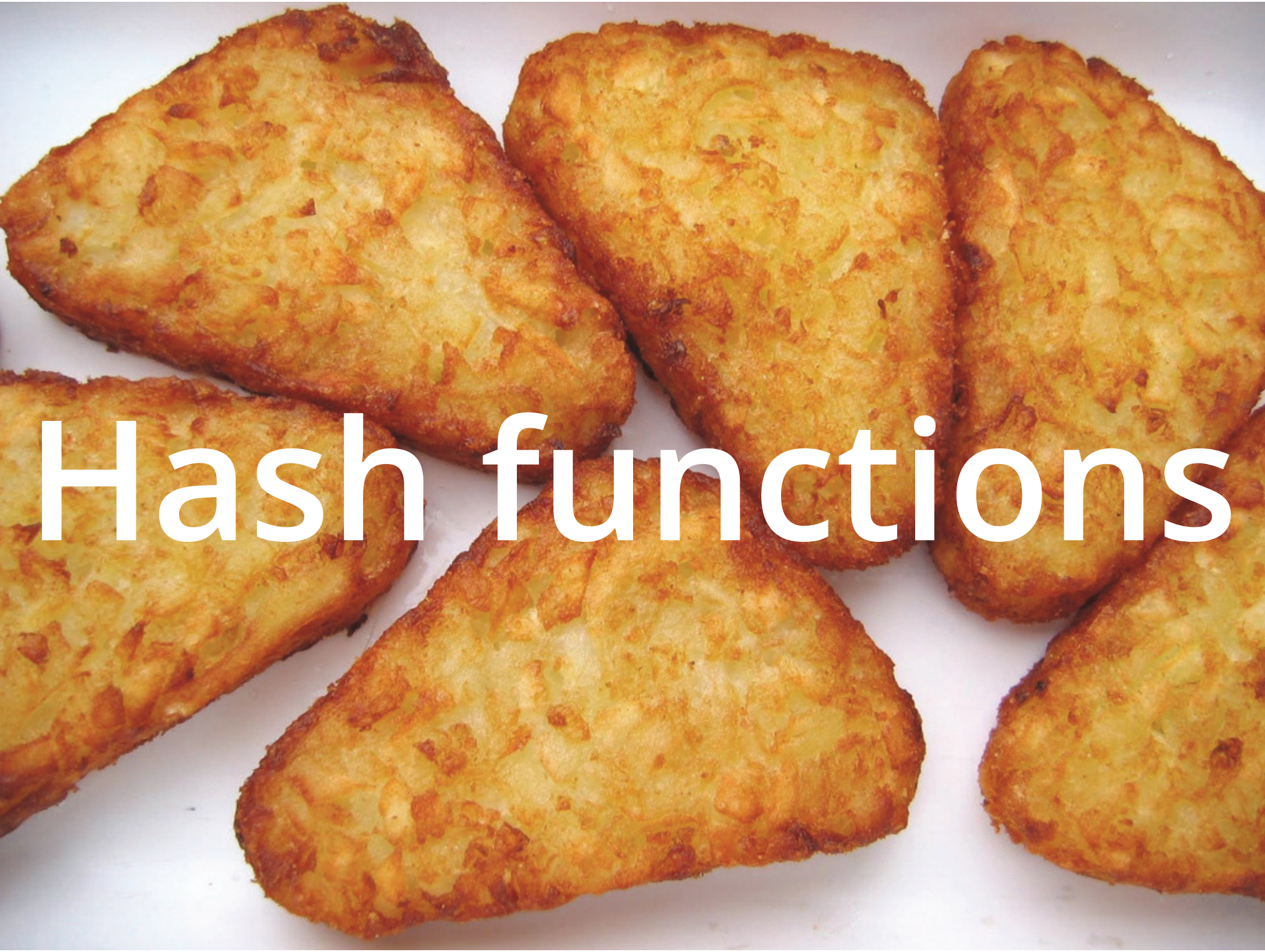
The Paris Kanellakis Theory and Practice Award honors specific theoretical accomplishments that have had a significant and demonstrable effect on the practice of computing. This award is accompanied by a prize of $10,000 and is endowed by contributions from the Kanellakis family, with additional financial support provided by ACM's Special Interest Groups on Algorithms and Computational Theory (SIGACT), Design Automaton (SIGDA), Management of Data (SIGMOD), and Programming Languages (SIGPLAN), the ACM SIG Projects Fund, and individual contributions.

Andrei Broder, Moses Charikar, Piotr Indyk Named Recipients Of The 2012 Paris Kanellakis Theory And Practice Award

Broder, Charikar, and Indyk were recognized for their work on algorithms that allow for quickly finding similar entries in large databases, known as locality-sensitive hashing (LSH), These algorithms can drastically reduce the computational time needed for retrieving similar items, at the cost of a small probability of failing to find the absolute closest match. LSH has impacted fields as diverse as computer vision, databases, information retrieval, data mining, machine learning, and signal processing.

Carnegie Mellon University

- Sparse Aggregators
  (Bloom, CountMin)
- Optimizing over Sparse Aggregators
  (Linear models, CoFi rank)
- Random function classes
  (Random Kitchen Sinks, Fast Food)
- Random Projections
  (LSH, SimHash, FastEx)

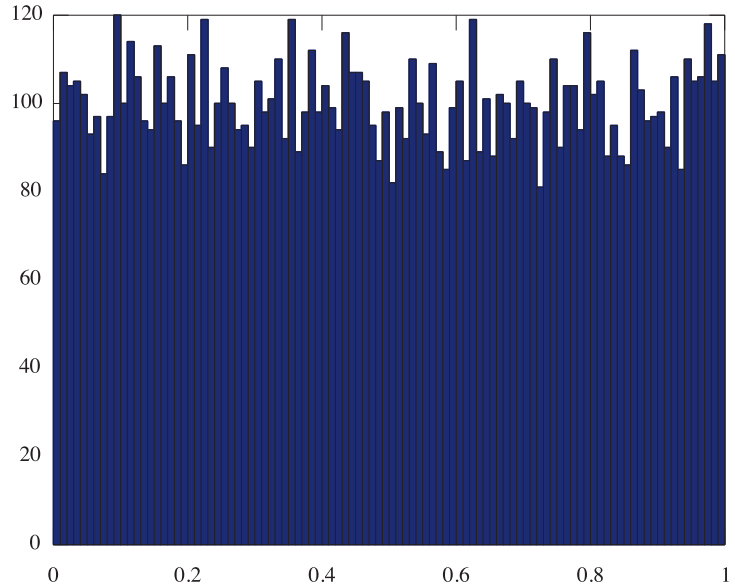

Hash functions

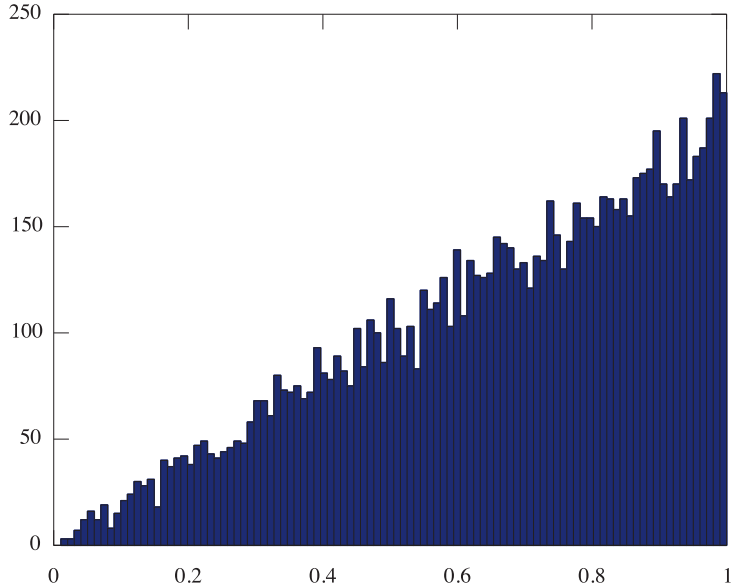
Random Number Generators

- In the beginning there was the uniform U[0,1]
  (10,000 draws from it)
- We use it for generating any other distribution
- Example
  Triangle distribution via

$$x \sim U[0,1] \text{ and } x \to \sqrt{x}$$

- Do we need to store this?





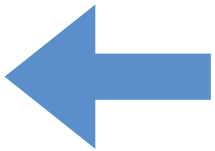
Random Number Generators

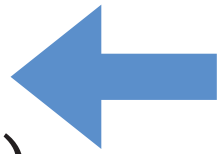
- Simple idea
  - Random number generator uses seed
  - Parametrize random numbers by seeds
  - No need to store hash values, only remember the seed key.
- Hash function

$$h : \mathcal{X} \to \{1 \ldots N\} \text{ where } \Pr\{h(x) = j\} = \frac{1}{N} \text{ for all } j \in \{1 \ldots N\}$$

Any set of keys x (e.g. text, images, users)

```
octave:1> rand(1)
ans =  0.082484
octave:2> rand(1)
ans =  0.99447
octave:3> rand("seed", 42)
octave:4> rand(1)
ans =  0.31198
octave:5> rand(1)
ans =  0.10358
octave:6> rand("seed", 2013)
octave:7> rand(1)
ans =  0.98535
octave:8> rand("seed", 42)
octave:9> rand(1)
ans =  0.31198
octave:10> rand(1)
ans =  0.10358
```



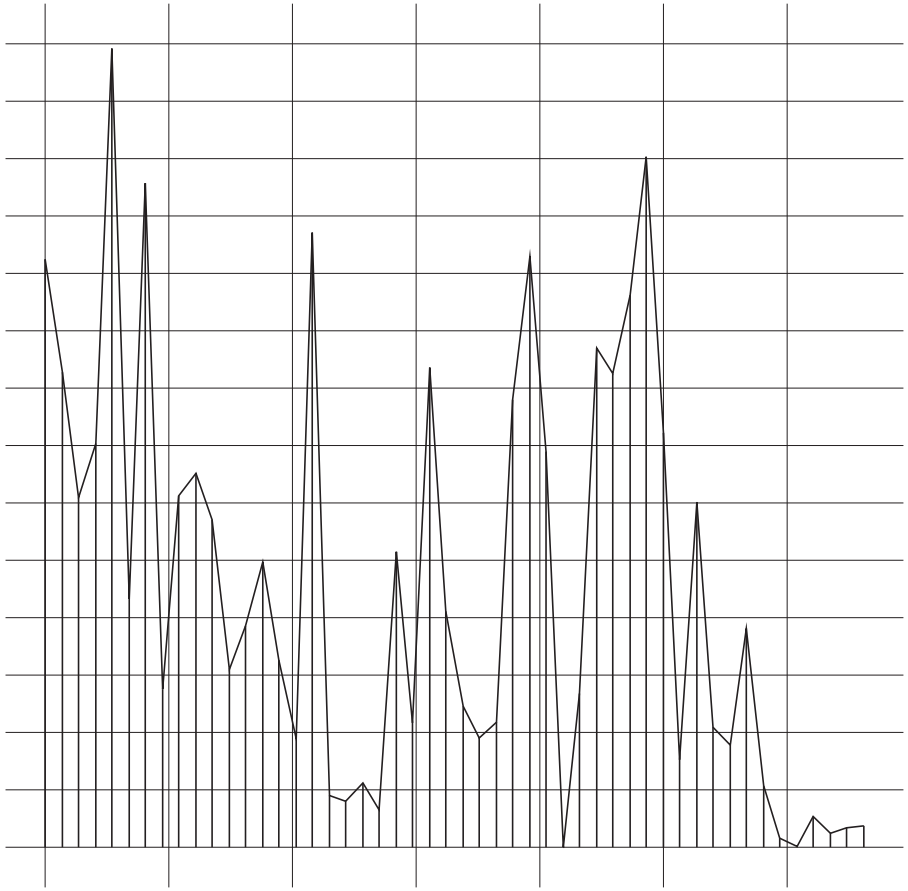


Hash Functions

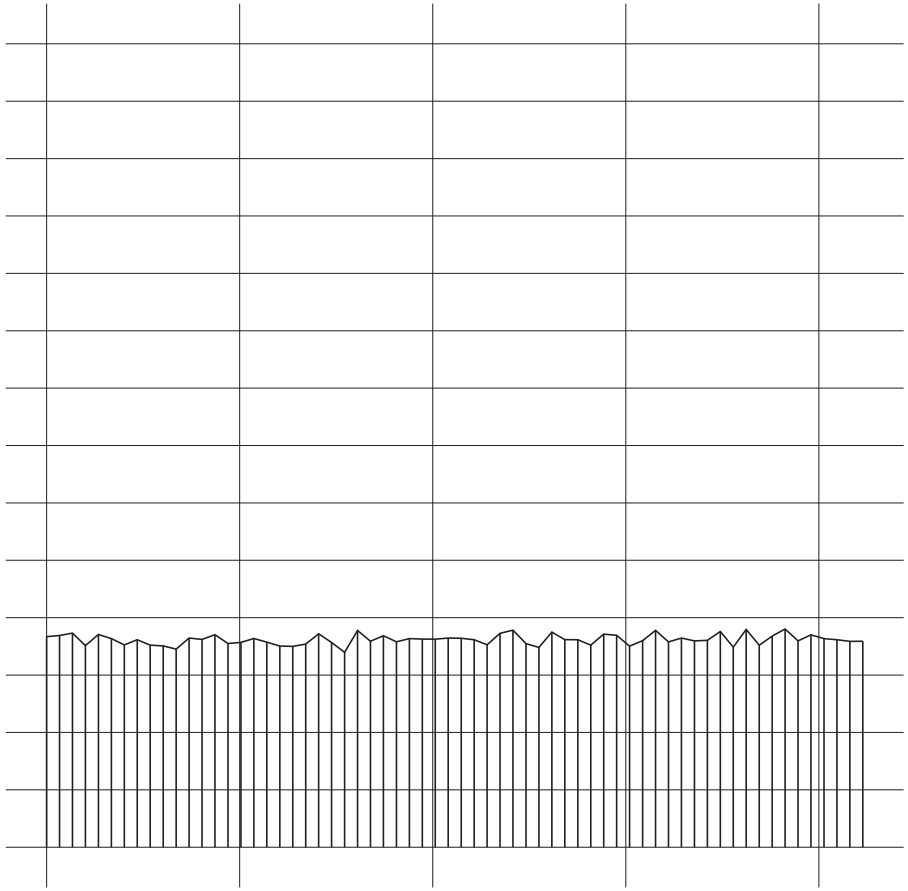
- Maps arbitrary data to uniform distribution
- Typically hash functions have MANY bits (e.g. used for file signatures via MD5 hash)

```
octave:20> md5sum('Institute',opt)
ans = 4d9ba0c70be930cc598307b366390e84
octave:21> md5sum('of',opt)
ans = 8bf8854bebe108183caeb845c7676ae4
octave:22> md5sum('Mathematics',opt)
ans = 540b21ecdb276f5087ee585cedd6d5d0
octave:23> md5sum('University of Minnessota',opt)
ans = 9450937cf3b656d9c93f92b22e79b128
octave:24> md5sum(99,opt)
ans = 4a8a08f09d37b73795649038408b5f33
```

Hash Functions

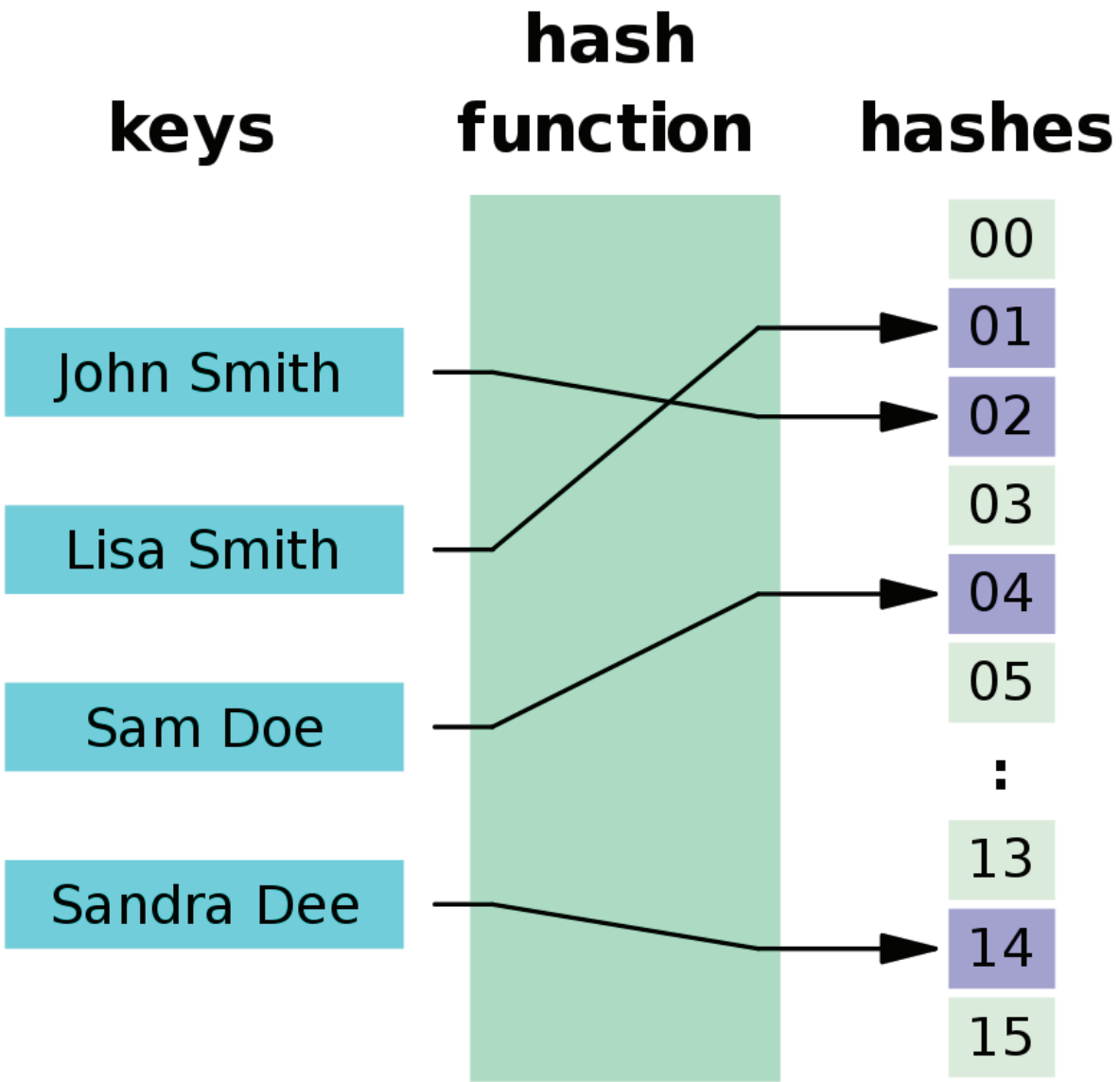


non-hashed



hashed

Hashing Strings



Independence

- **Very sloppy basic requirement**

$$h : \mathcal{X} \to \{1 \ldots N\} \text{ where } \Pr\{h(x) = j\} = \frac{1}{N}$$

  statement is over all hash functions

- **2-ways independence**
  Often good enough but makes proofs harder

$$\Pr\{h(x) = n \text{ and } h(x') = n'\} = N^{-2} \text{ for } x \neq x'$$

Independence

- **Very sloppy basic requirement**

$$h : \mathcal{X} \to \{1 \dots N\} \text{ where } \Pr\{h(x) = j\} = \frac{1}{N}$$

  statement is over all hash functions

- **2-ways independence**
  Often good enough but makes proofs harder

$$\Pr\{h(x) = n \text{ and } h(x') = n'\} = N^{-2} \text{ for } x \neq x'$$

- **k-ways independence**

$$\Pr\{h(x_1) = n_1 \dots h(x_k) = n_j\} = N^{-k}$$

Independence

- Very sloppy basic requirement

$$h : \mathcal{X} \to \{1 \dots N\} \text{ where } \Pr\{h(x) = j\} = \frac{1}{N}$$

  statement is over all hash functions

- 2-ways independence
  Often good enough but makes proofs harder

$$\Pr\{h(x) = n \text{ and } h(x') = n'\} = N^{-2} \text{ for } x \neq x'$$

- In the following we assume perfect hash function unless stated otherwise.

Application - checksums

- Alex and Bin have a file
  - Want to check whether Bin has the latest version of the slides.
  - Don't send the file unless it's different
- Alex and Bin compute (MD5) hash of file
  - `a = h(alex.pdf)  b = h(bin.pdf)`
  - Compare whether `a=b`
  - Send file only if not the same
- Probability of failure is 2^{-bits}
- Does not depend on the size of the file

```
alexsmola-macbookair2:talks alexsmola$ md5 ima.key
MD5 (ima.key) = eba6b9e730c427f8ddae4466ef69ec74
```


Streams

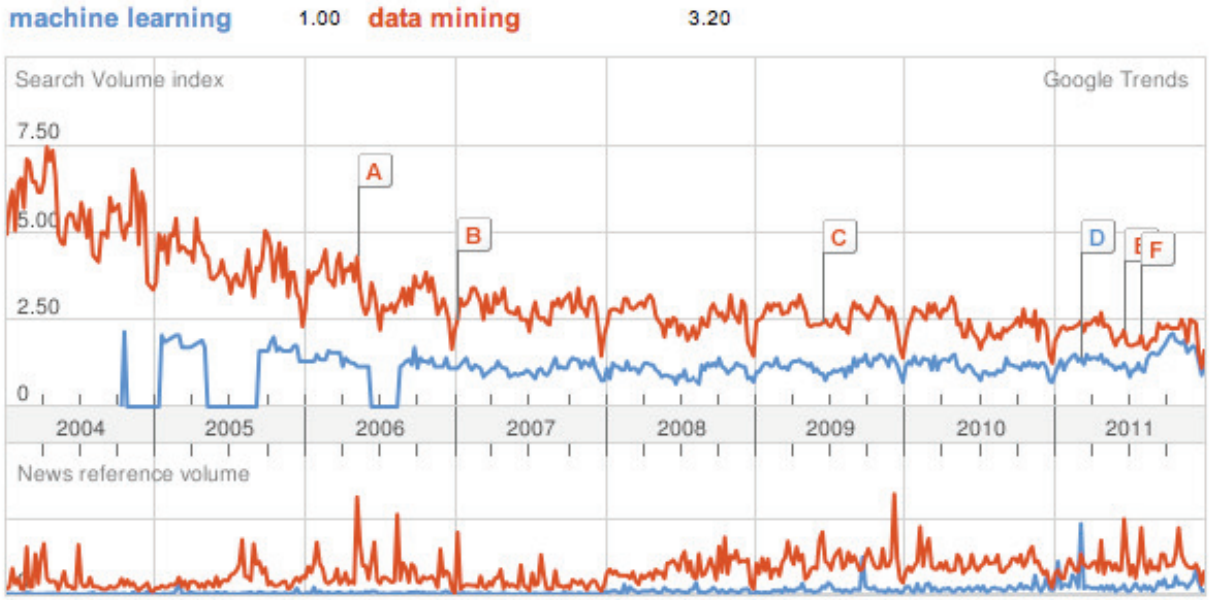
Aggregating Data Streams

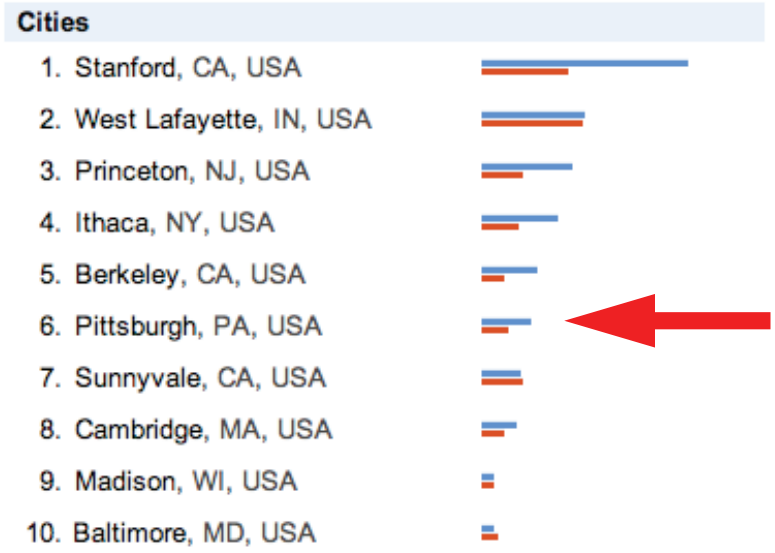
- Cannot replay data
  (e.g. logging in an MCMC sampler)
- Limited memory / computation / realtime analytics
- Time series
  Stock symbols, acceleration data, video, server logs
- Cash register
  Queries, user activity, network traffic, revenue, clicks, page views, logins
- Turnstile
  Increments and decrements (e.g. users active)

Website Analytics

- Continuous stream of users (tracked with cookie)
- Many sites signed up for analytics service
- Find hot links / frequent users / click probability / right now

Query Stream





- Item stream
- Find heavy hitters
- Detect trends early (e.g. Obama bin Laden killed)
- Frequent combinations (cf. frequent items)
- Source distribution
- In real time

Network traffic analysis

- TCP/IP packets
- On switch with limited memory footprint
- Realtime analytics
- Busiest connections
- Trends
- Protocol-level data
- Distributed information gathering

Financial Time Series

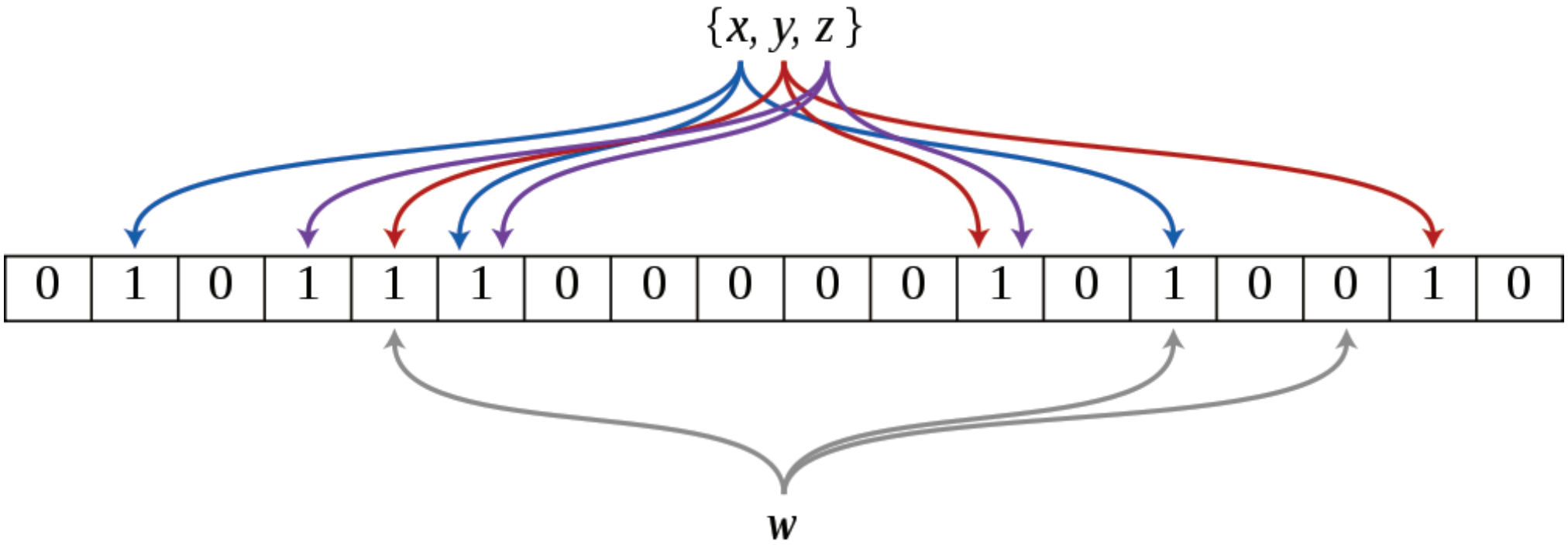
- time-stamped data stream
- multiple sources
- different time resolution

- real time prediction
- missing data
- metadata (news, quarterly reports, financial background)

Goals

- Check for previously seen items
  - Don’t need to have counts, just existence
- Check for frequency estimate
  - Don’t want to store labels
  - Want estimate for all items
  - Want to be able to aggregate easily
  - Want turnstile computation

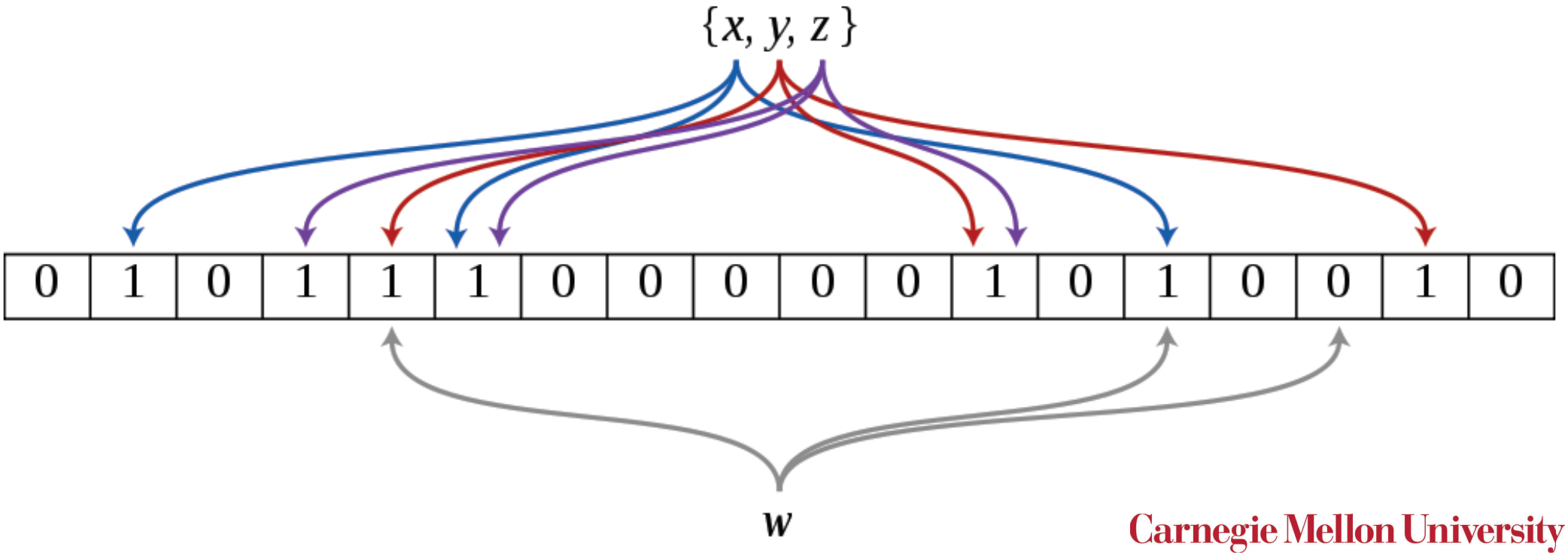
Bloom Filters



Bloom Filter

- **Bit array b of length n**

```
insert(x)
    for all i set bit b[h(x,i)]=1
query(x)
    return TRUE if for all i b[h(x,i)]=1
```



Bloom Filter

- Bit array b of length n
  insert(x): for all i set bit b[h(x,i)] = 1
  query(x): return TRUE if for all i b[h(x,i)] = 1
- Only returns TRUE if all k bits are set
- No false negatives but false positives possible
  - Probability that an arbitrary bit is set

$$\Pr\{b[i]=1\} = 1 - \left(1 - \frac{1}{n}\right)^{mk} \approx 1 - e^{-\frac{mk}{n}}$$

  - Probability of false positive (approx. indep.)

$$\Pr\{b[h(x,1)] = \ldots = b[h(x,k)] = 1\} \approx \left(1 - e^{-\frac{mk}{n}}\right)^{k}$$

Bloom Filter

- Minimizing k to minimize false positive rate

$$\partial_k \left[k \log \left(1 - e^{-mk/n} \right) \right] = \log \left(1 - e^{-mk/n} \right) + \frac{mk}{n} \frac{e^{-mk/n}}{1 - e^{-mk/n}}$$

  This vanishes for $\frac{mk}{n} = \log 2$ and hence $k = \frac{n}{m} \log 2$ with a false positive rate of 2^{-k}

- More refined analysis & details, e.g. in the Mitzenmacher & Broder tutorial.

Broder, Andrei; Mitzenmacher, Michael (2005), "Network Applications of Bloom Filters: A Survey", *Internet Mathematics* **1** (4): 485–509,

- Matching lower bound shows that Bloom filter is within 1.44 best efficiency.

Cool things to do with a Bloom Filter

- Bloom filter of union of two sets by OR

0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	1	1	1

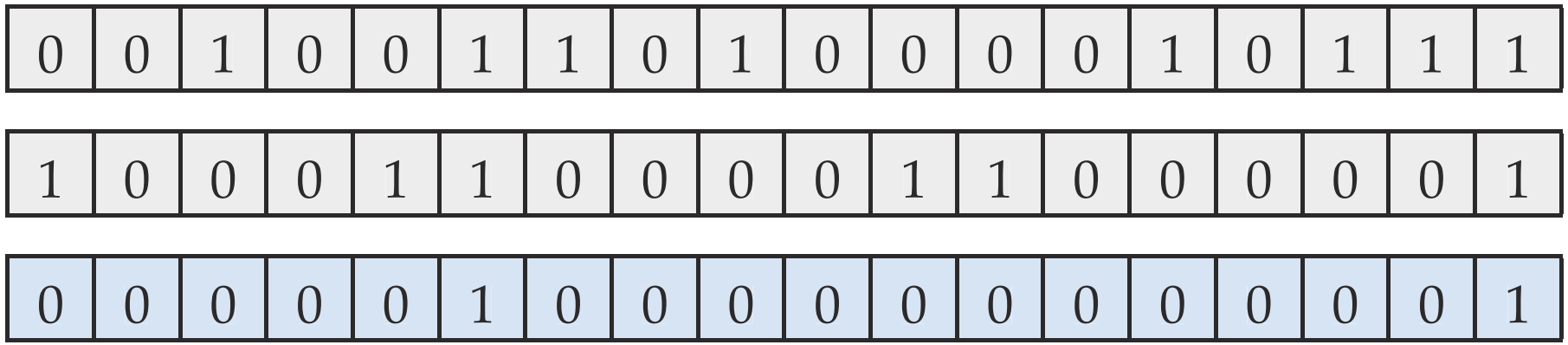
1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	1

1	0	1	0	1	1	1	0	1	0	1	1	0	1	0	1	1	1

  - Parallel construction of Bloom filters
  - Time-dependent aggregation
  - Fast approximate set union
    (bitmap operation rather than set manipulation)
- Also use it to halve bit resolution of Bloom filter

Cool things to do with a Bloom Filter

- Set intersection via AND



  - No false negatives
  - More false positives than building from scratch
- Use bits to estimate size of set union/intersection

$$\Pr\{b=1\} = \Pr\{b=1|S_1\} + \Pr\{b=1|S_2\} - \Pr\{b=1|S_1 \cup S_2\}$$

$$\approx 1 - e^{-\frac{k|S_1|}{m}} - e^{-\frac{k|S_2|}{m}} + e^{-\frac{k|S_1 \cup S_2|}{m}}$$

Counting Bloom Filter

- Plain Bloom filter doesn't allow removal

0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	1	1	1

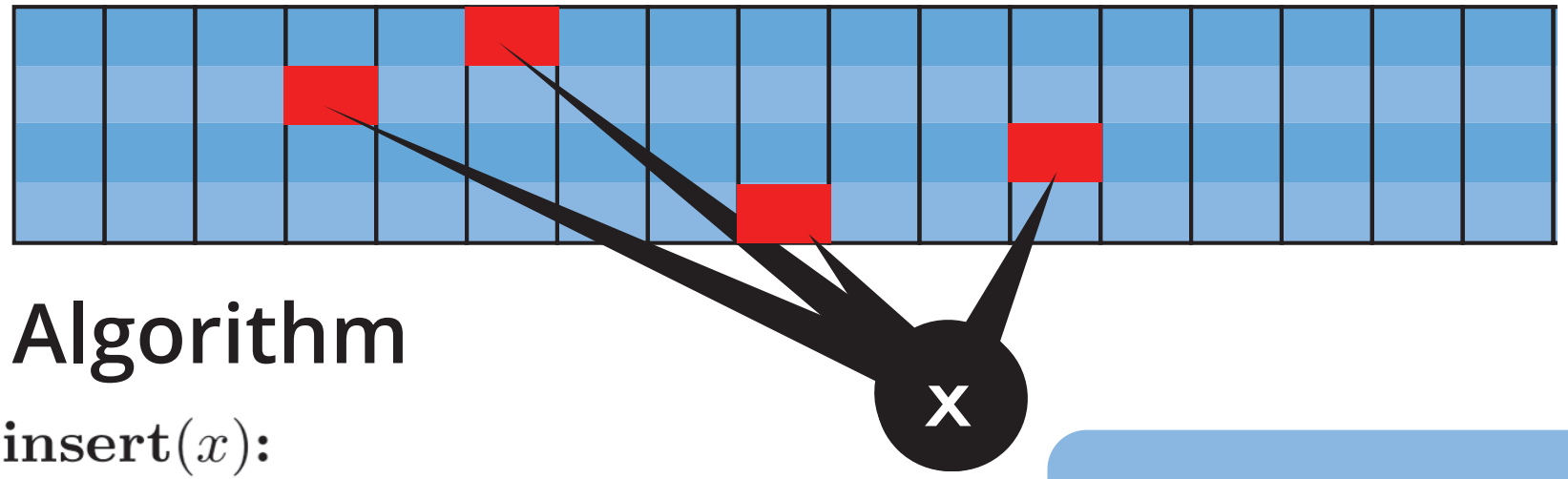
  - insert(x): for all i set bit b[h(x,i)] = 1
    we don't know whether this was set before
  - query(x): return TRUE if for all i b[h(x,i)] = 1
- Counting Bloom filter keeps track of inserts
  - query(x): return TRUE if for all i b[h(x,i)] > 0
  - insert(x): if query(x) = FALSE (don't insert twice)
    for all i increment b[h(x,i)] = b[h(x,i)] + 1
  - remove(x): if query(x) = TRUE (don't remove absents)
    for all i decrement b[h(x,i)] = b[h(x,i)] - 1

only needs
log log m bits


Count min sketch

Count min sketch

- Datastructure



- Algorithm

insert(x)**:**
for $i = 1$ **to** d **do**
$M[i, h_i(x)] \leftarrow M[i, h_i(x)] + 1$
end for

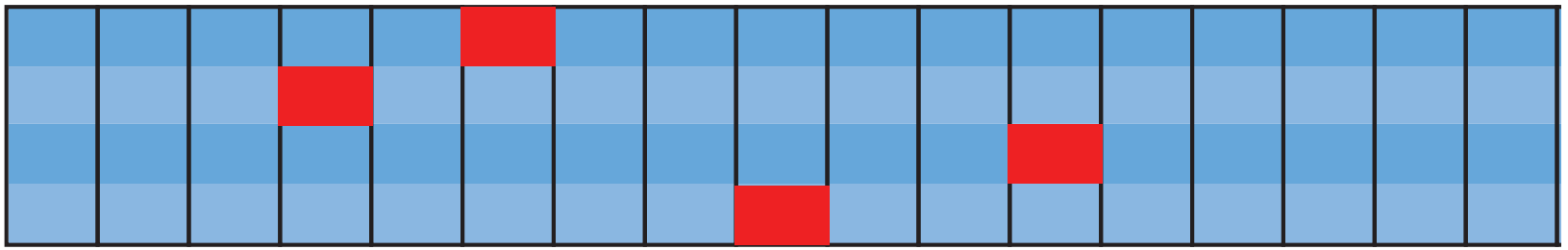
query(x)**:**
$c = \min\{h_i(x) \text{ for all } 1 \leq i \leq d\}$
return c

like Bloom filter but with counters

supports turnstile

Count min sketch

- Datastructure



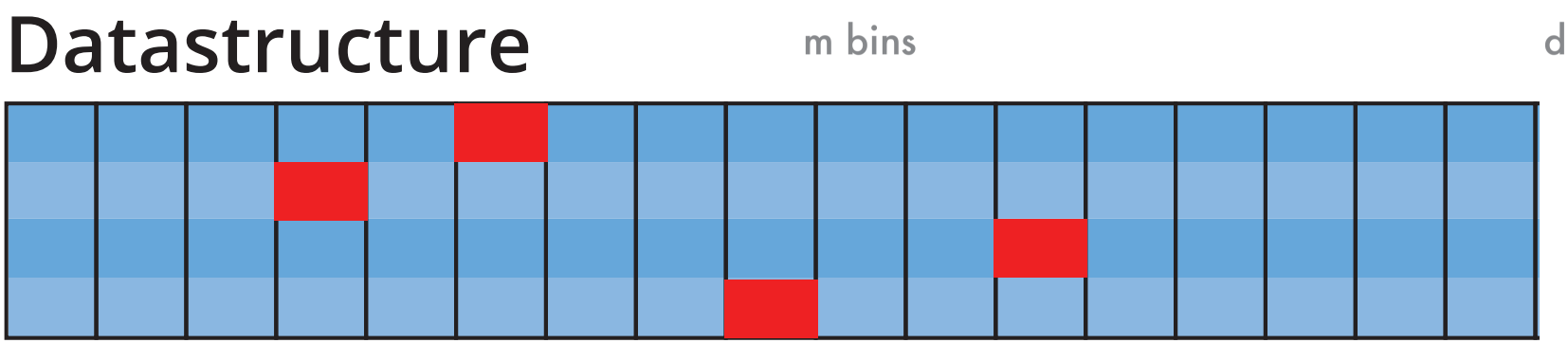
- Guarantees
  - Approximation quality is

$$n_x \leq c_x \leq n_x + \epsilon \sum_{x'} n_{x'} \text{ for } m = \left\lceil \frac{e}{\epsilon} \right\rceil \text{ with probability } 1 - e^{-d}$$

  - For power law distributions with exponent z we need only $O(\epsilon^{-1/z})$ space (see Cormode & Muthukrishnan, 2004)

Proof

- Datastructure



- Lower bound
  - Each bin is updated whenever we see an item
  - So each bin is lower bound, hence min is OK
- Expectation
  - Probability of incrementing a bin at random is 1/m, hence expected overestimate is n/m.

Proof

- **Gauss-Markov inequality on random variable**

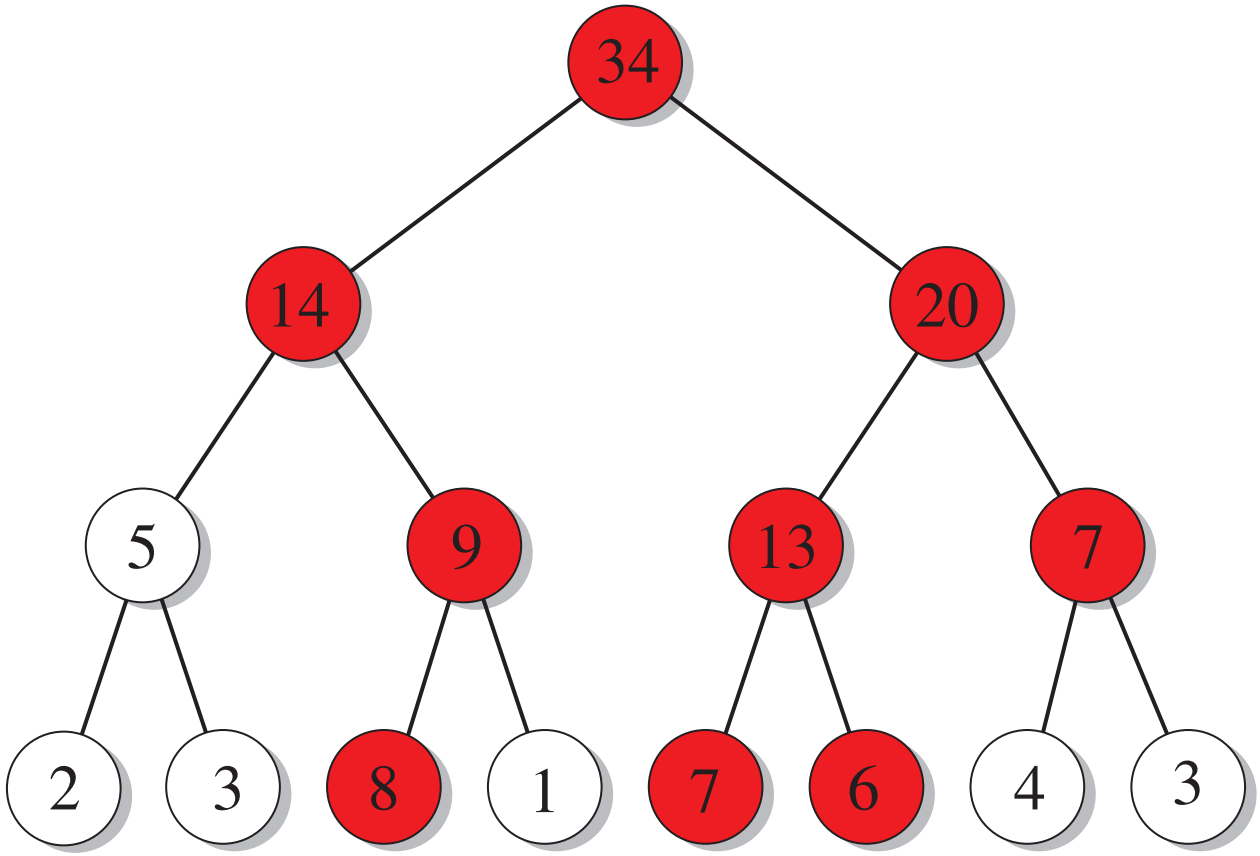
$$\mathbf{E}\left[w[i,h(i,x)]-n_x\right]=\frac{n}{m} \text{ hence } \Pr\left\{w[i,h(i,x)]-n_x>e\frac{n}{m}\right\}\leq e^{-1}$$

- **Minimum boosts probability exponentially (only need to ensure that there's at least one random variable which satisfies the condition)**

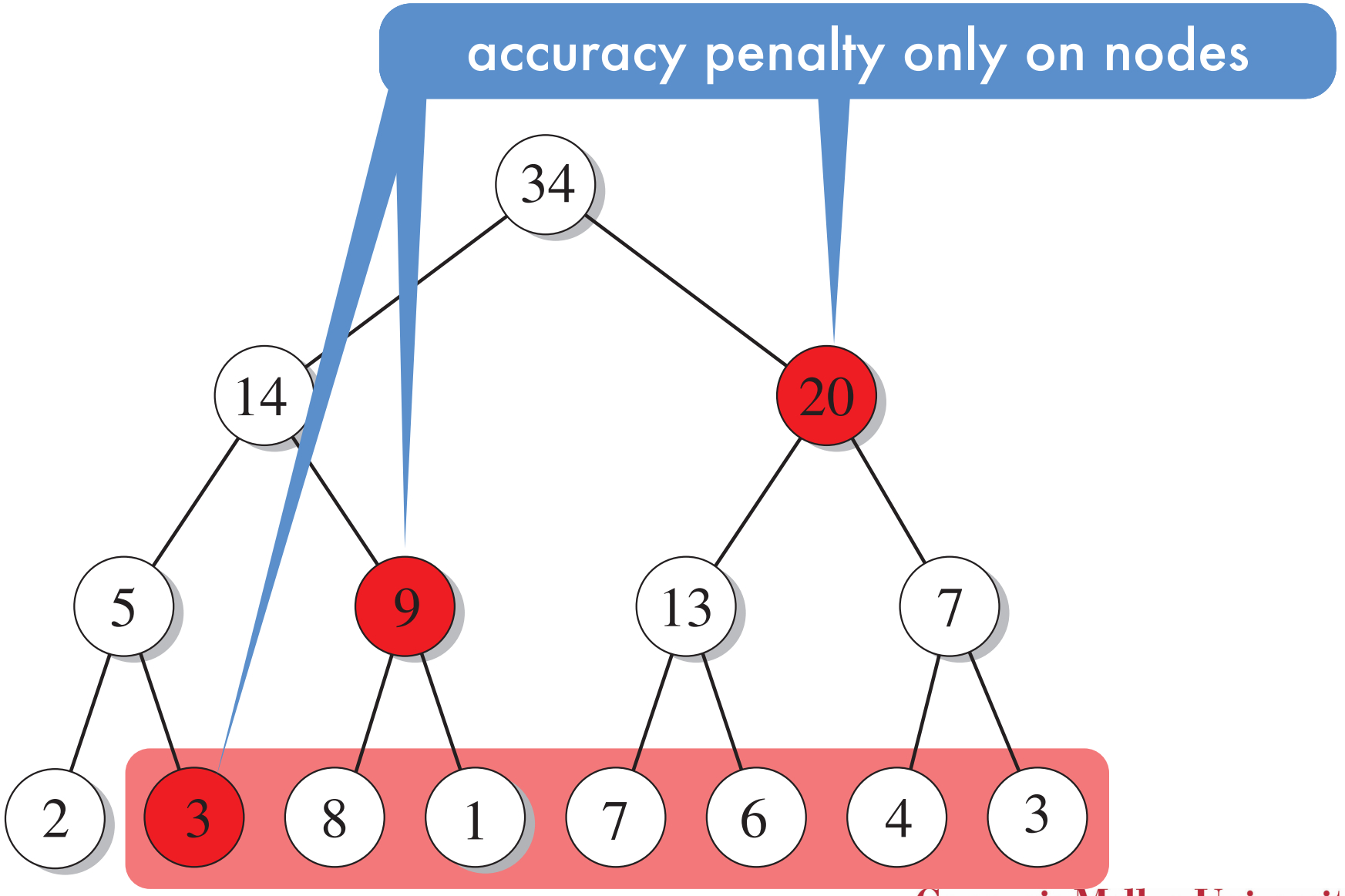
$$\Pr\left\{c_x-n_x>e\frac{n}{m}\right\}\leq e^{-d}$$

Heavy Hitters finding

- Hierarchical event structure
  - IP numbers
  - Prices
  - Activity logs
- Keep top nodes explicitly
- Traverse range via CM sketch



Range query



Tail guarantees

- **Zipfian distributions**

$$\Pr\{x\} = \frac{c}{(a+x)^z}$$

- **Bounding heads/tails (for a = 0 and z > 1)**

$$c_z k^{1-z} \leq \sum_{i=k}^{\infty} f_i \leq c_z (k-1)^{1-z}$$

  - only small number of heavy items exists
  - bound heavy hitters separately
  - probability of collision is small
  - tail is small enough for low offset

Tail guarantees

- Set head to m/3 of all bins
- Probability we don't hit head is 2/3 per hash

$$\mathbf{E}[c_x|\text{noheavy}] = n_x + \frac{3}{2m}\sum_{i=k+1, i\neq x}^{\infty} n_i \leq n_x + c_z\frac{n^{-z}}{m} \text{ for } k = \frac{n}{3}$$

- Apply Gauss-Markov for 'noheavy' with p=1/2
- Boost residual probability by min operation
- The space needed for Zipfian distribution is $O\left(\epsilon^{-\min\{1,1/z\}}\log 1/\delta\right)$ with $\Pr\{c_x > n_x + \epsilon n\} \leq \delta$

Counter Braids

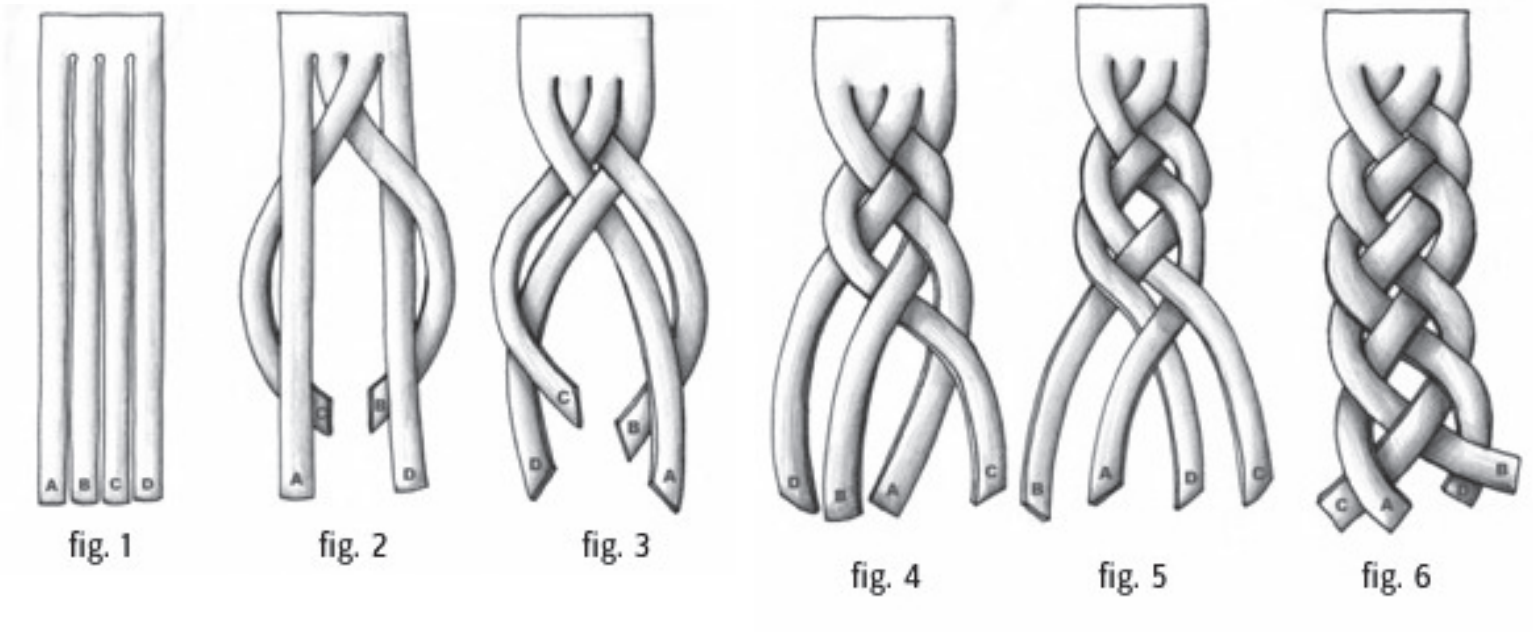
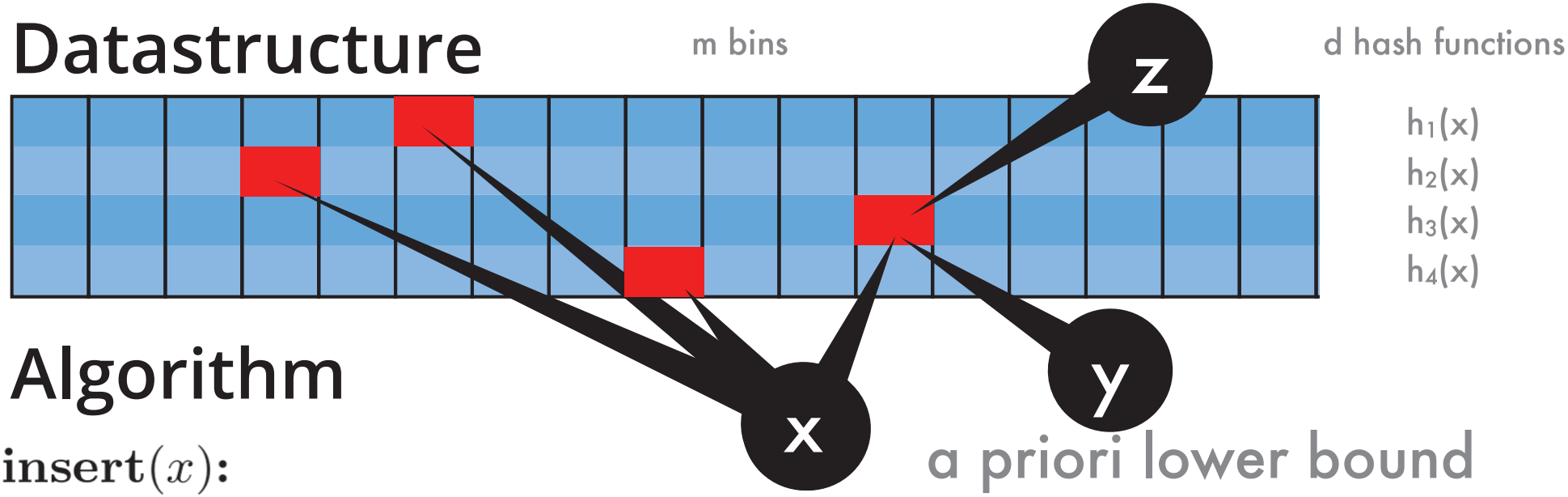


fig. 1 fig. 2 fig. 3 fig. 4 fig. 5 fig. 6

Part A - The Counter

- Datastructure



- Algorithm

insert(x)**:**
for $i = 1$ **to** d **do**
$M[i, h_i(x)] \leftarrow M[i, h_i(x)] + 1$
end for

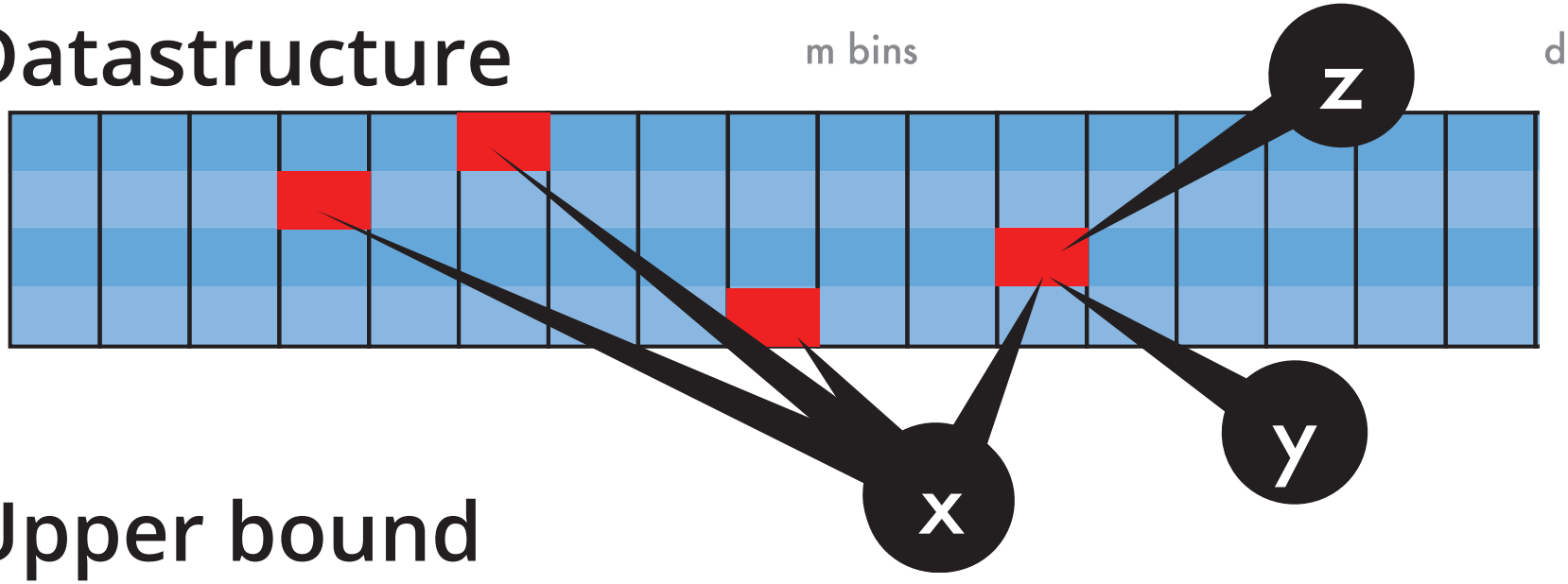
query(x)**:**
$c = \min\{h_i(x) \text{ for all } 1 \leq i \leq d\}$
return c

a priori lower bound on counter is 0

if we know all inserts we can get new lower bound

Part A - The Counter

- **Datastructure**



- **Upper bound**

$$w[i,j] = \sum_{h(i,x)=j} n_x \leq \sum_{h(i,x)=j} c_x \text{ hence } c_x \geq l_x := w[i,j] - \sum_{h(i,x)=j, x' \neq x} c_{x'}$$

- **Lower bound**

$$w[i,j] \geq \sum_{h(i,x)=j} l_x \text{ hence } c_x \leq u_x := w[i,j] - \sum_{h(i,x)=j, x' \neq x} l_{x'}$$

Part A - The Counter

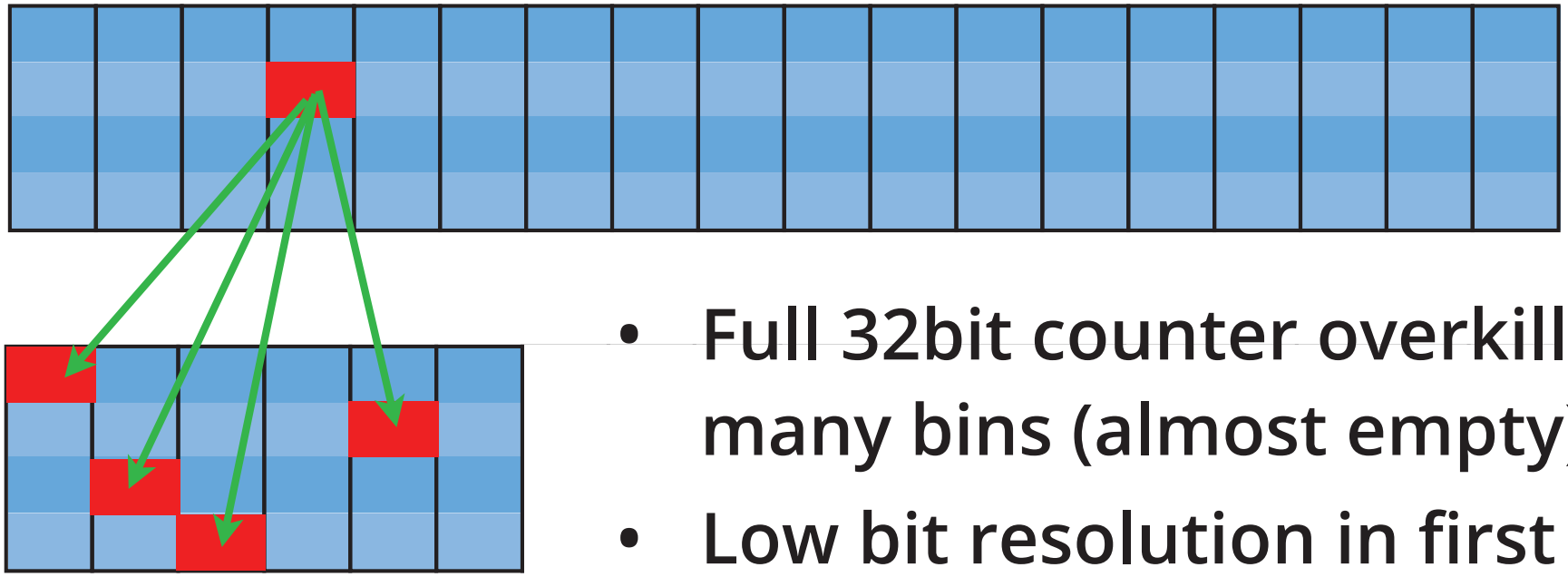
- Iterate lower & upper bounds 'til converged
  - proof highly nontrivial
  - cheap construction, expensive decoding
- Lower bound

$$w[i,j] = \sum_{h(i,x)=j} n_x \leq \sum_{h(i,x)=j} c_x \text{ hence } c_x \geq l_x := w[i,j] - \sum_{h(i,x)=j, x' \neq x} c_{x'}$$

- Upper bound

$$w[i,j] \geq \sum_{h(i,x)=j} l_x \text{ hence } c_x \leq u_x := w[i,j] - \sum_{h(i,x)=j, x' \neq x} l_{x'}$$

Part B - The Braid



- Full 32bit counter overkill for many bins (almost empty)
- Low bit resolution in first filter
- Insert overflows into secondary counter
- Cascade filters
- Reconstruction by iteration

http://newport.eecs.uci.edu/~athina/netsys/08/slides/balaji-braids.pdf

- Sparse Aggregators
  (Bloom, CountMin)
- **Optimizing over Sparse Aggregators**
  (Linear models, CoFi rank)
- Random function classes
  (Random Kitchen Sinks, Fast Food)
- Random Projections
  (LSH, SimHash, FastEx)

Hash Kernels & Linear Models

Vector Space Model for Text

A Vector Space Model for Automatic Indexing

G. Salton, A. Wong
and C. S. Yang
Cornell University

Communications
of
the ACM

November 1975
Volume 18
Number 11

In a document retrieval, or other pattern matching environment where stored entities (documents) are compared with each other or with incoming patterns (search requests), it appears that the best indexing (property) space is one where each entity lies as far away from the others as possible; in these circumstances the value of an indexing system may be expressible as a function of the density of the object space; in particular, retrieval performance may correlate inversely with space density. An approach based on space density computations is used to choose an optimum indexing vocabulary for a collection of documents. Typical evaluation results are shown, demonstating the usefulness of the model.

Linear functions

- Function

$$f(x) = \langle w, x \rangle + b = \sum_i w_i x_i + b$$

- Used for spam filtering, internet advertising, ranking, retrieval, summarization, gene finding, stock prediction, user profiling, …
- Good as long as we don't have more than 1 billion parameters (8GB of memory for `double` precision)
- What if we have more parameters?
  - Lasso (with distributed optimization)
  - Hashing

Personalized Spam Filtering

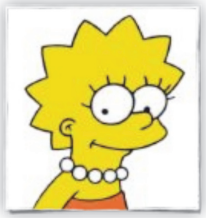
From: bat <kilian@gmail.com>
Subject: **hey whats up check this meds place out**
Date: April 6, 2009 10:50:13 PM PDT
To: Kilian Weinberger
Reply-To: bat <kilian@gmail.com>

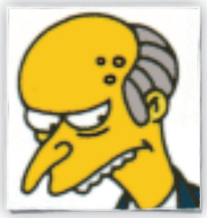
Your friend (kilian@gmail.com) has sent you a link to the following Scout.com story:
Savage Hall Ground-Breaking Celebration

Get Vicodin, Valium, Xanax, Viagra, Oxycontin, and much more. Absolutely No Prescription Required. Over Night Shipping! Why should you be risking dealing with shady people. Check us out today!
http://jenkinstegar73.blogspot.com

The University of Toledo will hold a ground-breaking celebration to kick-off the UT Athletics Complex and Savage Hall renovation project on Wednesday, December 12th at Savage Hall.

To read the rest of this story, go here:
http://toledo.scout.com/2/708390.html




Personalized Spam Filtering

From: bat <kilian@gmail.com>
Subject: **hey whats up check this meds place out**
Date: April 6, 2009 10:50:13 PM PDT
To: Kilian Weinberger
Reply-To: bat <kilian@gmail.com>

Your friend (kilian@gmail.com) has sent you a link to the following Scout.com story:
Savage Hall Ground-Breaking Celebration

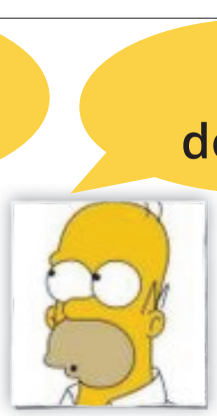
Get Vicodin, Valium, Xanax, Viagra, Oxycontin, and much more. Absolutely No Prescription Required. Over Night Shipping! Why should you be risking dealing with shady people. Check us out today!
http://jenkinstegar73.blogspot.com

The University of Toledo will hold a ground-breaking celebration to kick-off the UT Athletics Complex and Savage Hall renovation project on Wednesday, December 12th at Savage Hall.

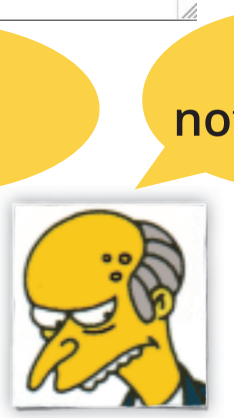
To read the rest of this story, go here:
http://toledo.scout.com/2/708390.html

educated

misinformed



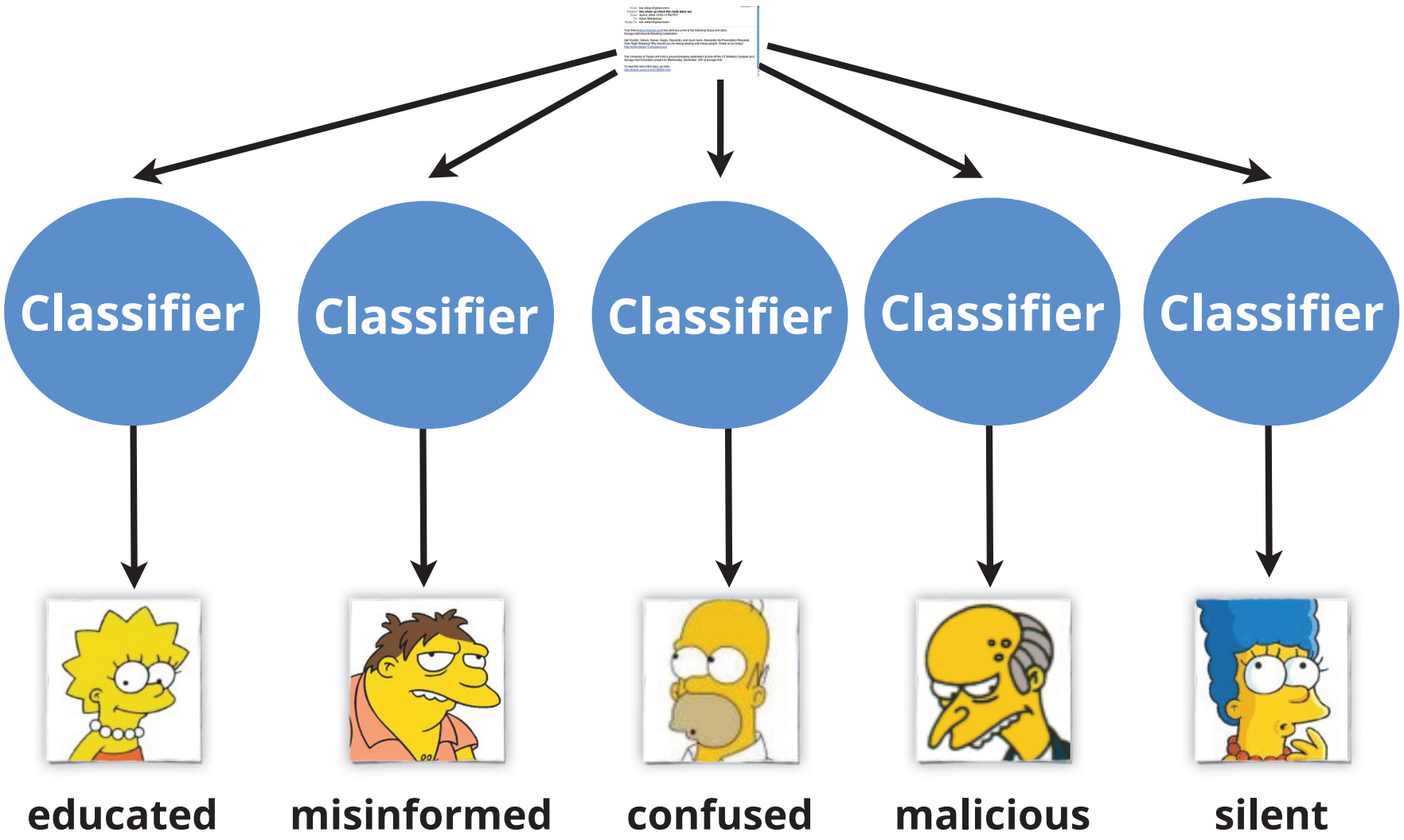
confused



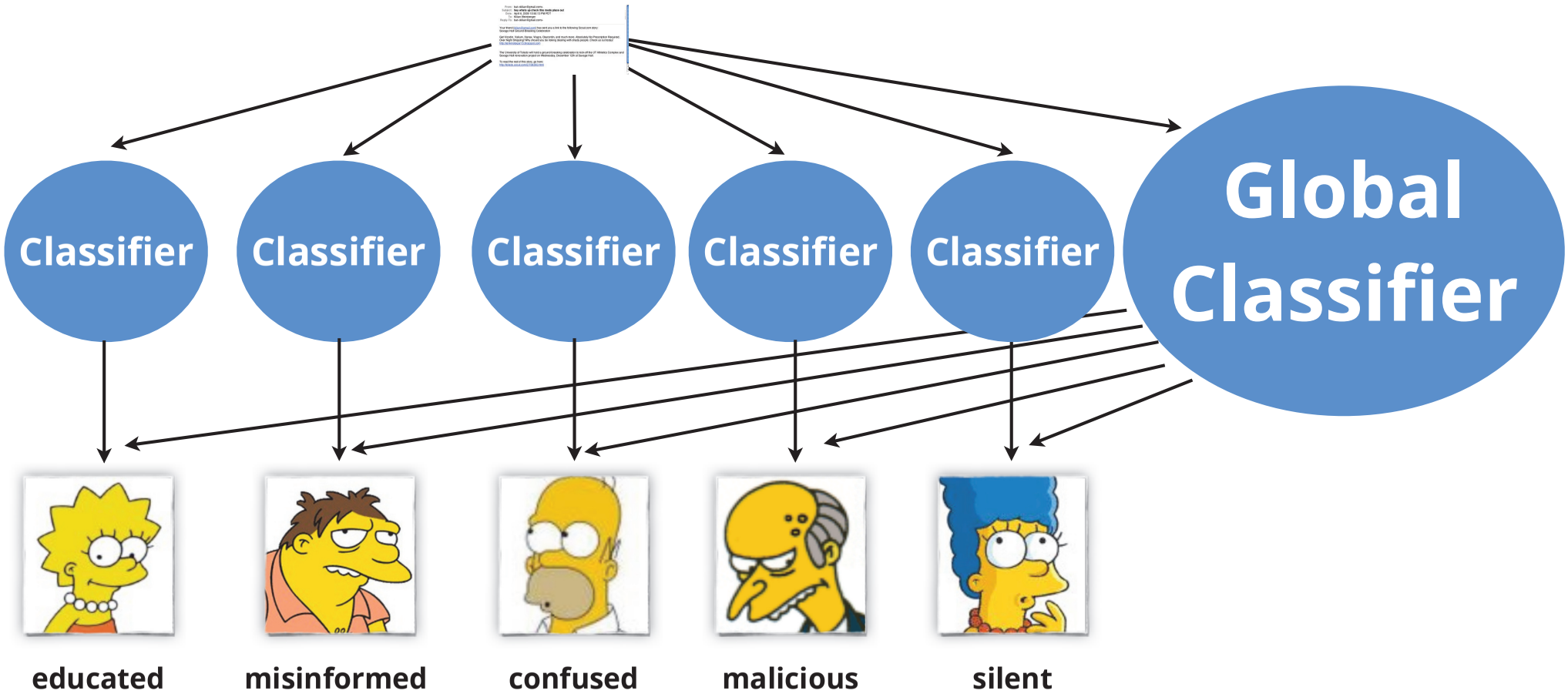
malicious

silent

Personalized Spam Filtering



Multitask Learning



Collaborative Classification

- **Primal space representation**

$$f(x,u) = \langle \phi(x), w \rangle + \langle \phi(x), w_u \rangle = \langle \phi(x) \otimes (1 \oplus e_u), w \rangle$$

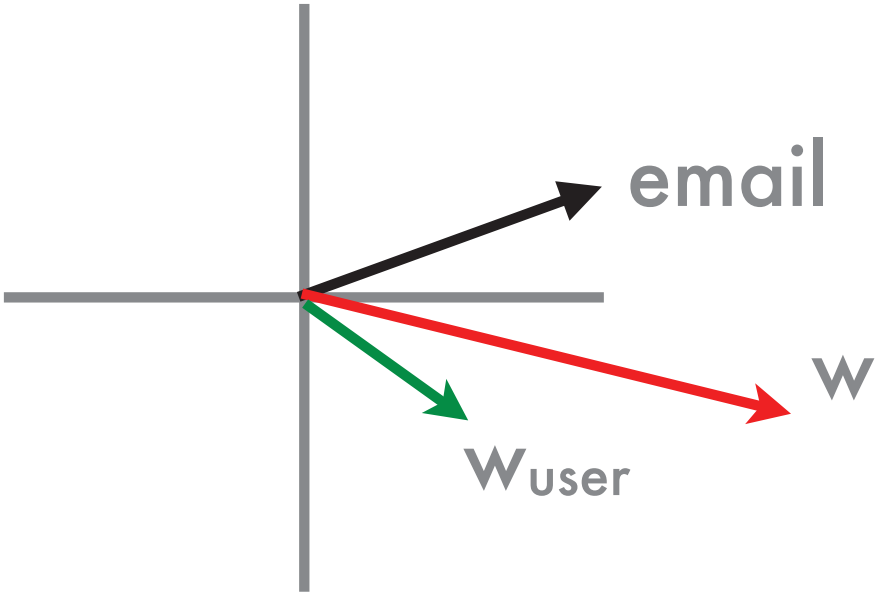
Kernel representation

$$k((x,u),(x',u')) = k(x,x')[1 + \delta_{u,u'}]$$

Multitask kernel (e.g. Pontil & Michelli, Daume). Usually does not scale well ...

- **Problem -** dimensionality is 10^{13}. That is 40TB of space

Collaborative Classification



- **Primal space representation**

$$f(x,u) = \langle \phi(x), w \rangle + \langle \phi(x), w_u \rangle = \langle \phi(x) \otimes (1 \oplus e_u), w \rangle$$

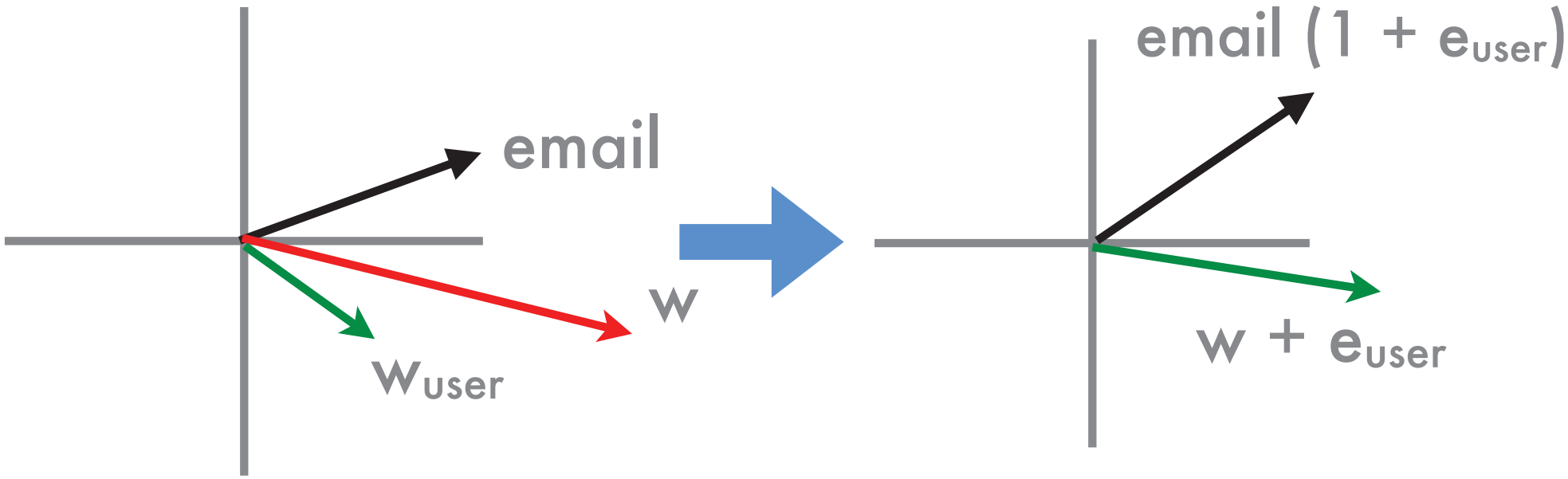
Kernel representation

$$k((x,u),(x',u')) = k(x,x')[1 + \delta_{u,u'}]$$

Multitask kernel (e.g. Pontil & Michelli, Daume). Usually does not scale well ...

- **Problem -** dimensionality is 10^{13}. That is 40TB of space

Collaborative Classification



- **Primal space representation**

$$f(x,u) = \langle \phi(x), w \rangle + \langle \phi(x), w_u \rangle = \langle \phi(x) \otimes (1 \oplus e_u), w \rangle$$

Kernel representation

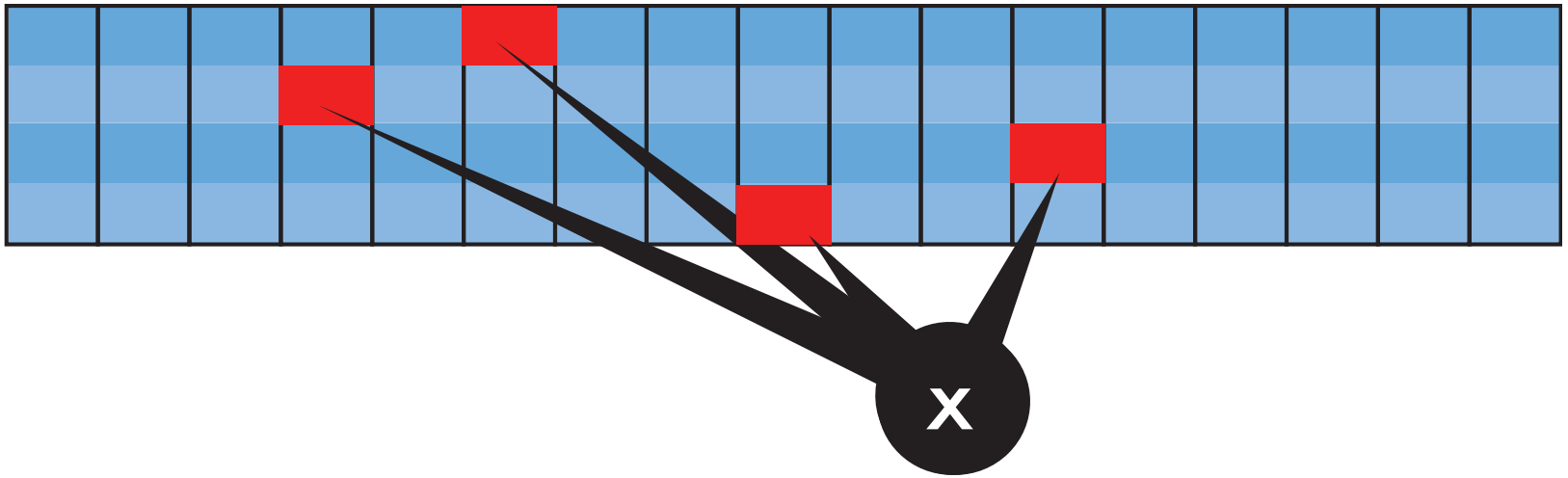
$$k((x,u),(x',u')) = k(x,x')[1 + \delta_{u,u'}]$$

Multitask kernel (e.g. Pontil & Michelli, Daume). Usually does not scale well ...

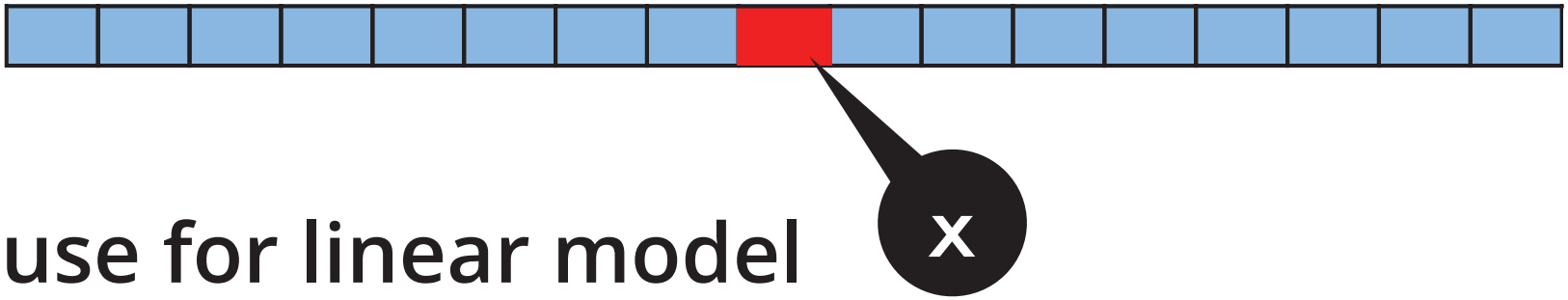
- **Problem -** dimensionality is 10^{13}. That is 40TB of space

CountMin Datastructure

- Count Sketch Basic



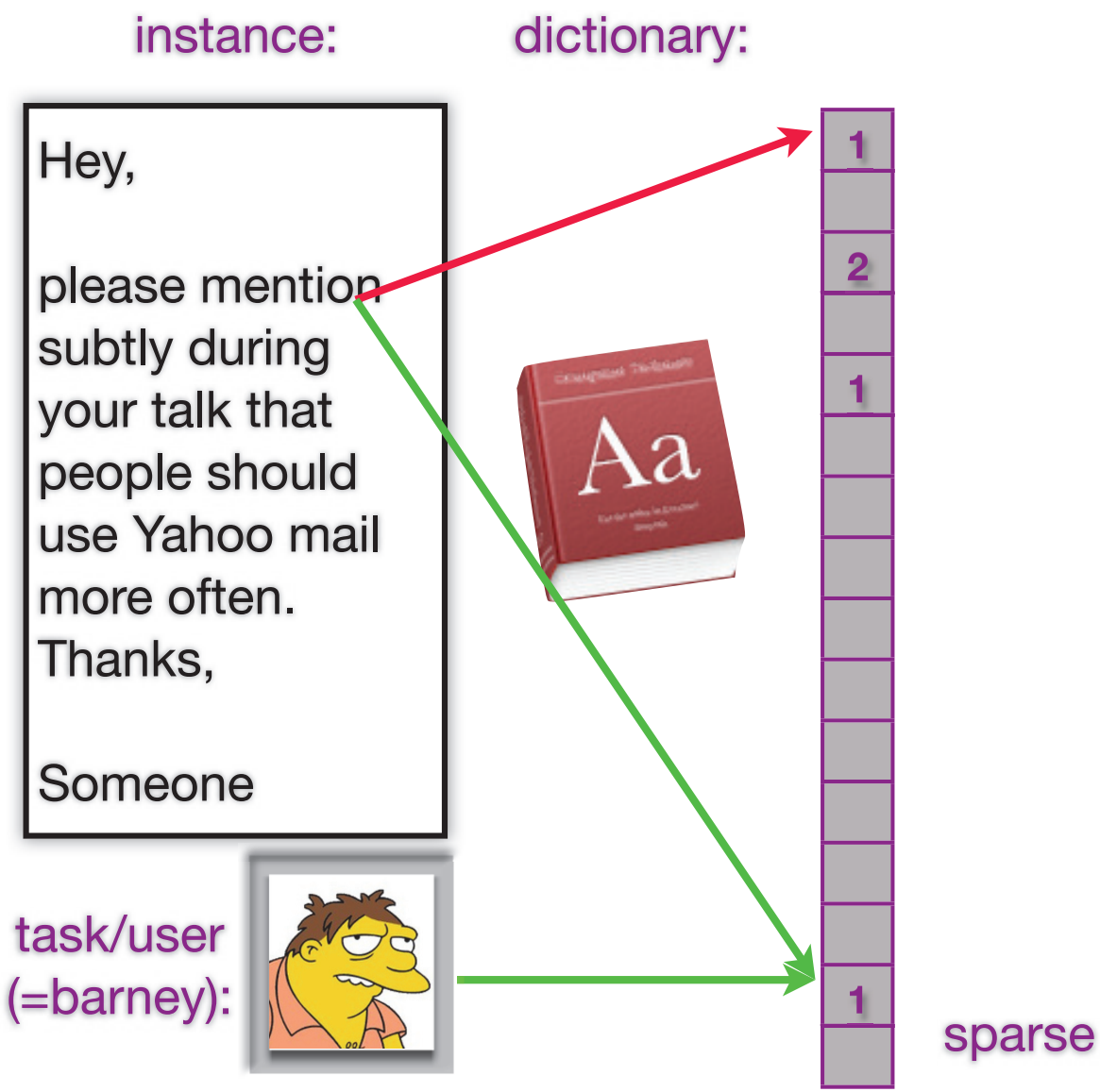
- Making it even more sparse



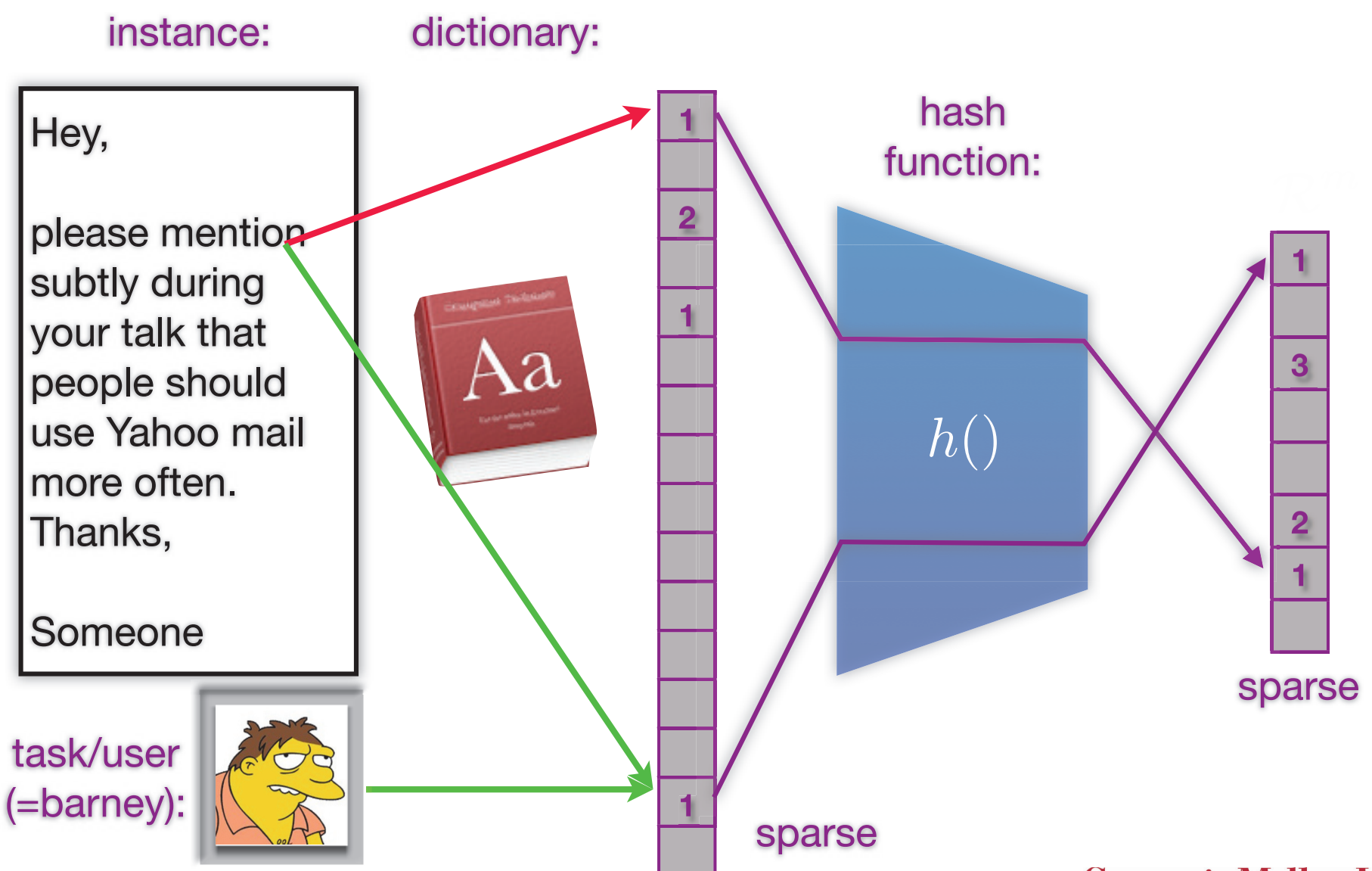
use for linear model

Hash Kernel

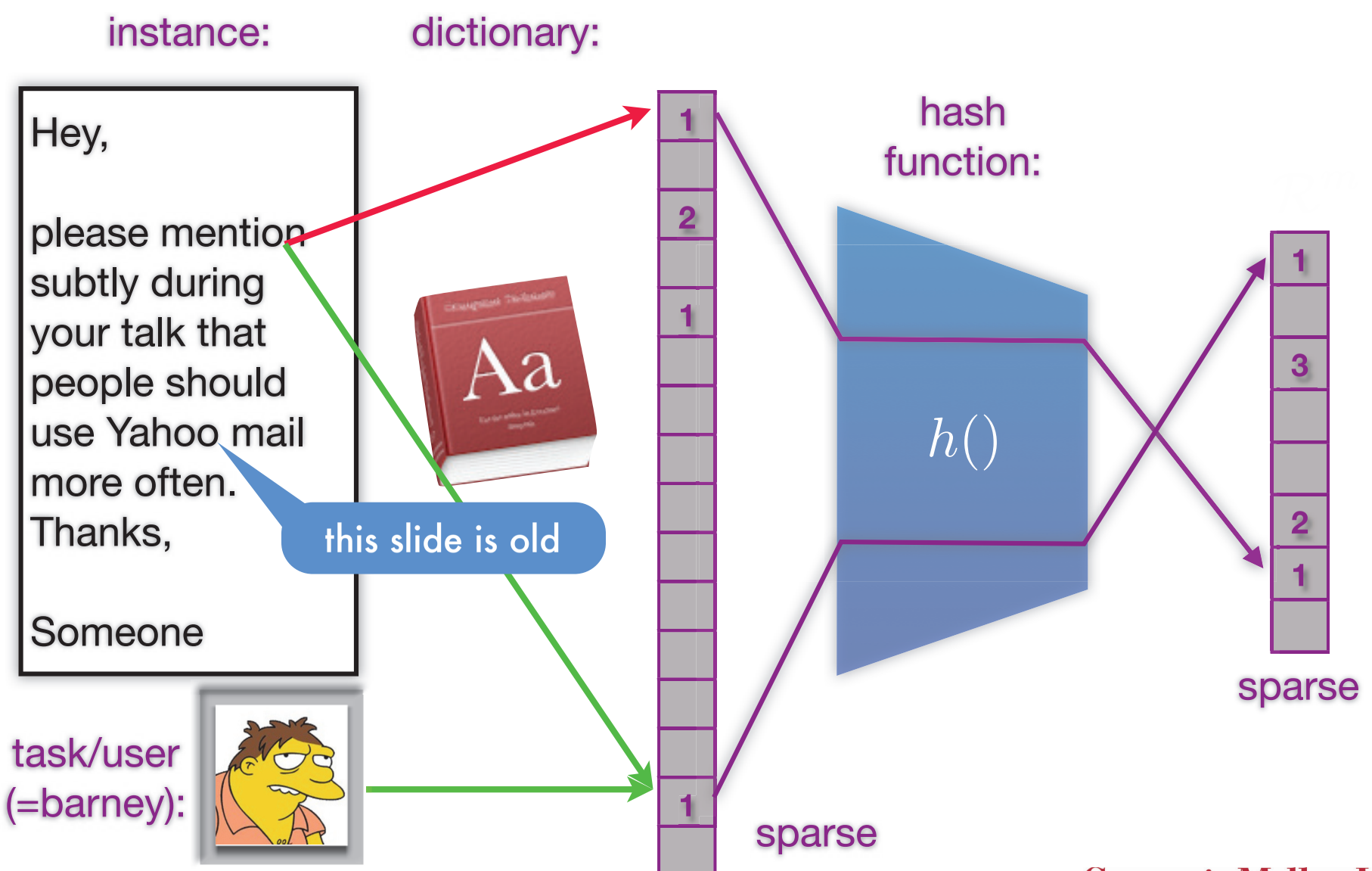
Hash Kernel


instance:
dictionary:
Hey,
please mention subtly during your talk that people should use Yahoo mail more often.
Thanks,
Someone
task/user (=barney):
1
2
1
1
sparse

Hash Kernel


instance:
dictionary:
Hey,
please mention subtly during your talk that people should use Yahoo mail more often.
Thanks,
Someone
task/user (=barney):
hash function:
$h()$
sparse
sparse

Hash Kernel


instance:
dictionary:
Hey,
please mention subtly during your talk that people should use Yahoo mail more often.
Thanks,
Someone
this slide is old
task/user (=barney):
hash function:
$h()$
sparse
sparse

Hash Kernel

instance:

Hey,

please mention subtly during your talk that people should use Yahoo mail more often.
Thanks,

Someone

task/user (=barney):

$$f(x) = \sum_i w[h(i)]\sigma(i)x_i$$

{-1, 1}

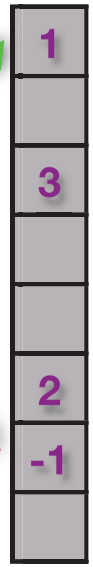
h('mention')

h('mention_barney')

$h()$

s(m_b)

s(m)



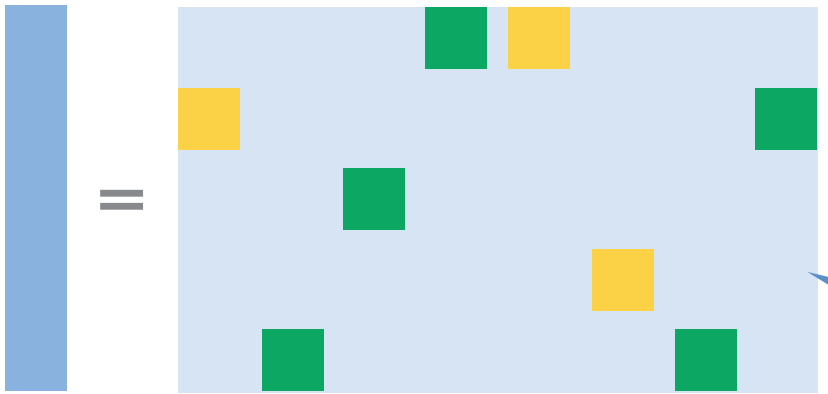
Similar to count hash
(Charikar, Chen, Farrach-Colton, 2003)

Advantages of hashing

- No dictionary!
  - Content drift is no problem
  - All memory used for classification
  - Finite memory guarantee (good for online learning)

- No Memory needed for projection (vs. LSH).
- Implicit mapping into high dimensional space!
- Sparsity preserving! (vs LSH)

Hash Kernels - the matrix view



$$= \quad \times \quad z = Px$$

maximally sparse matrix

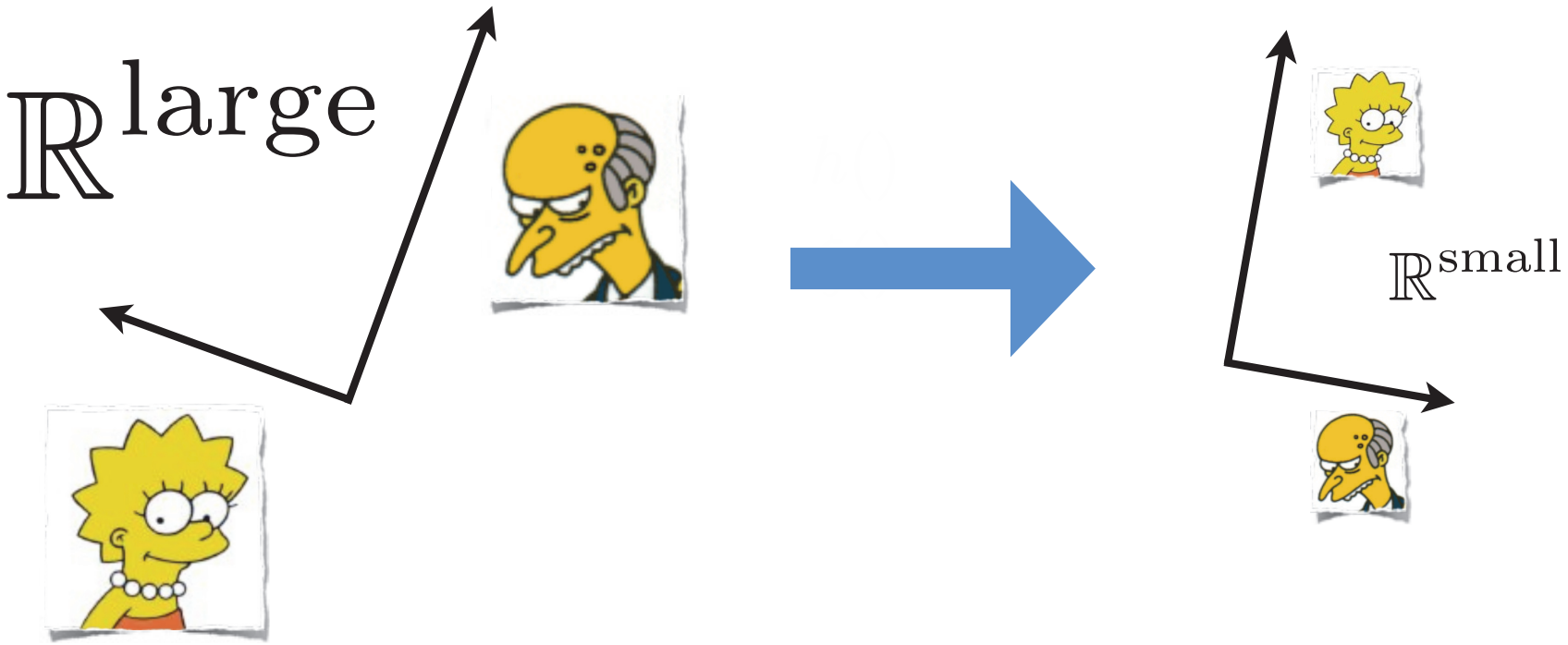
- Preserves inner product

$$\langle w, x \rangle = \sum_i w_i x_i \qquad \langle \bar{w}, \bar{x} \rangle = \sum_j \left[\sum_{i:h(i)=j} w_i \sigma(i) \right] \left[\sum_{i:h(i)=j} x_i \sigma(i) \right]$$

Rademacher hash

$$\mathbf{E}_\sigma[\sigma(i)\sigma(i')] = \delta_{ii'}$$

Approximate Orthogonality



We can do multi-task learning!

Guarantees

- For a random hash function the inner product vanishes with high probability via

$$\Pr\{|\langle w_v, h_u(x)\rangle| > \epsilon\} \leq 2e^{-C\epsilon^2 m}$$

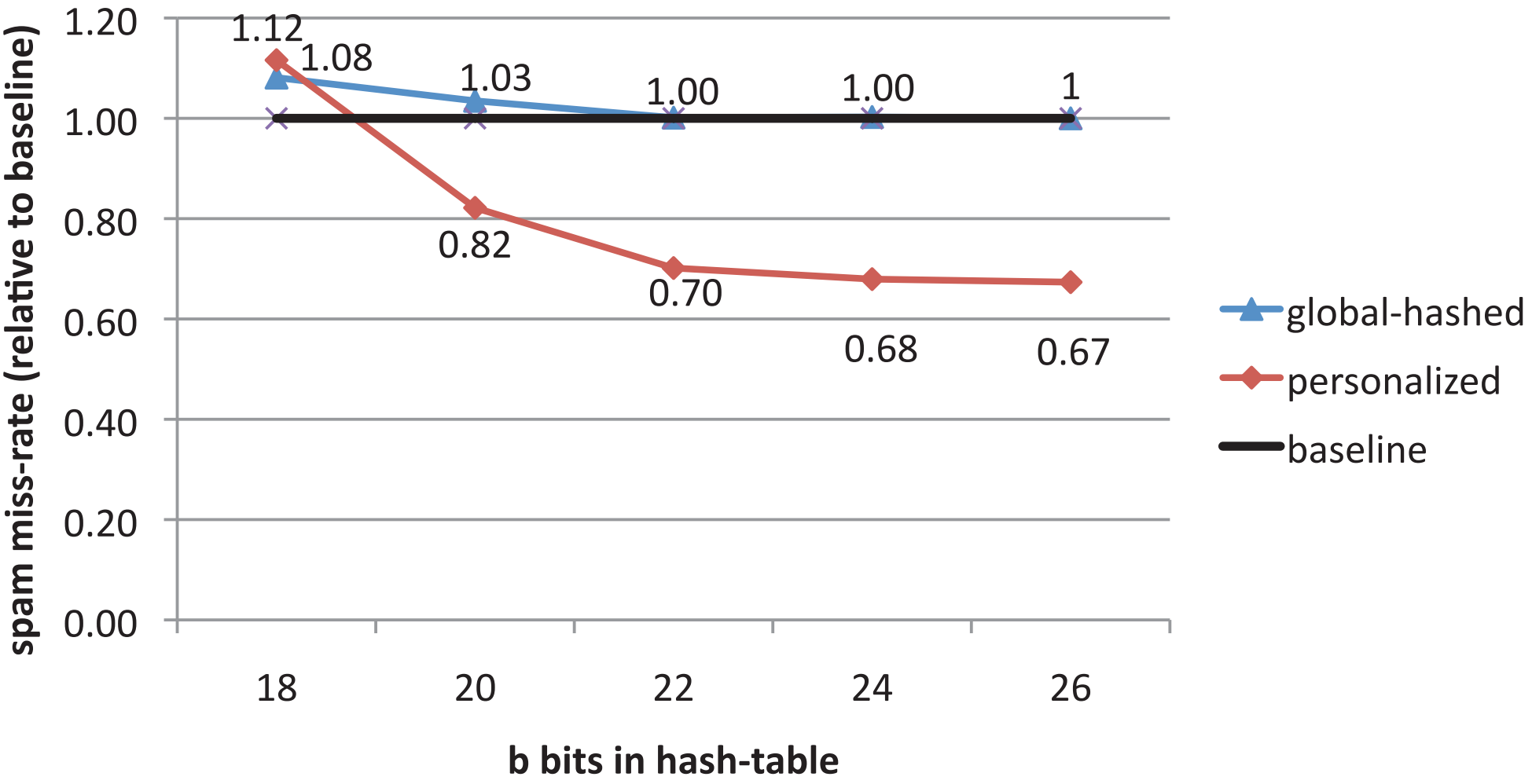
- We can use this for multitask learning

Direct sum in Hilbert Space → **Sum** in Hash Space

- Hashed inner product is unbiased
- Variance is O(1/n)
- Restricted isometry property (Kumar, Sarlos, Dasgupta 2010)

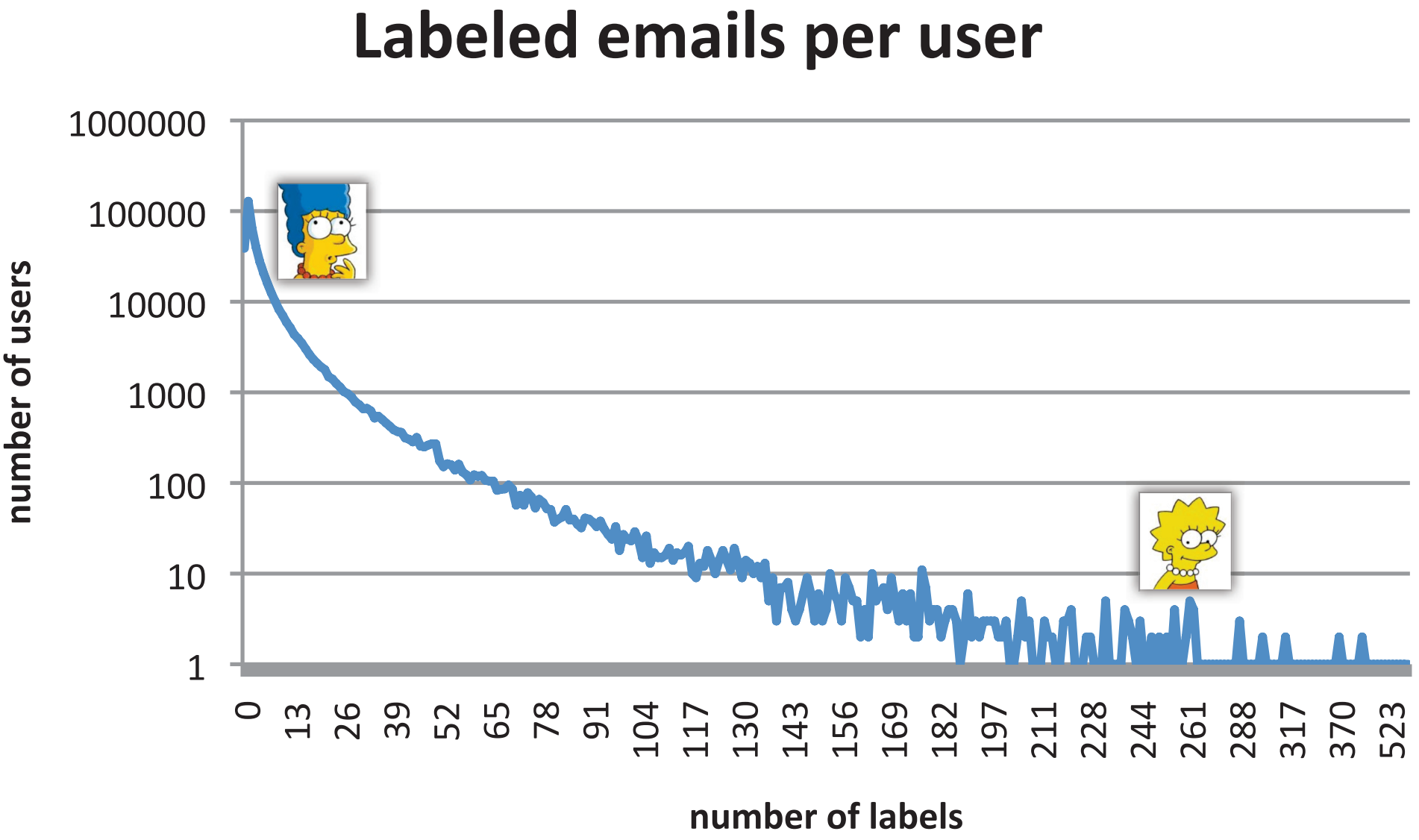
Weinberger, K., Dasgupta, A., Attenberg, J., Langford, J., and Smola, A. J.,Feature Hashing for Large Scale Multitask Learning, International Conference on Machine Learning, 2009. PDF

Spam classification results

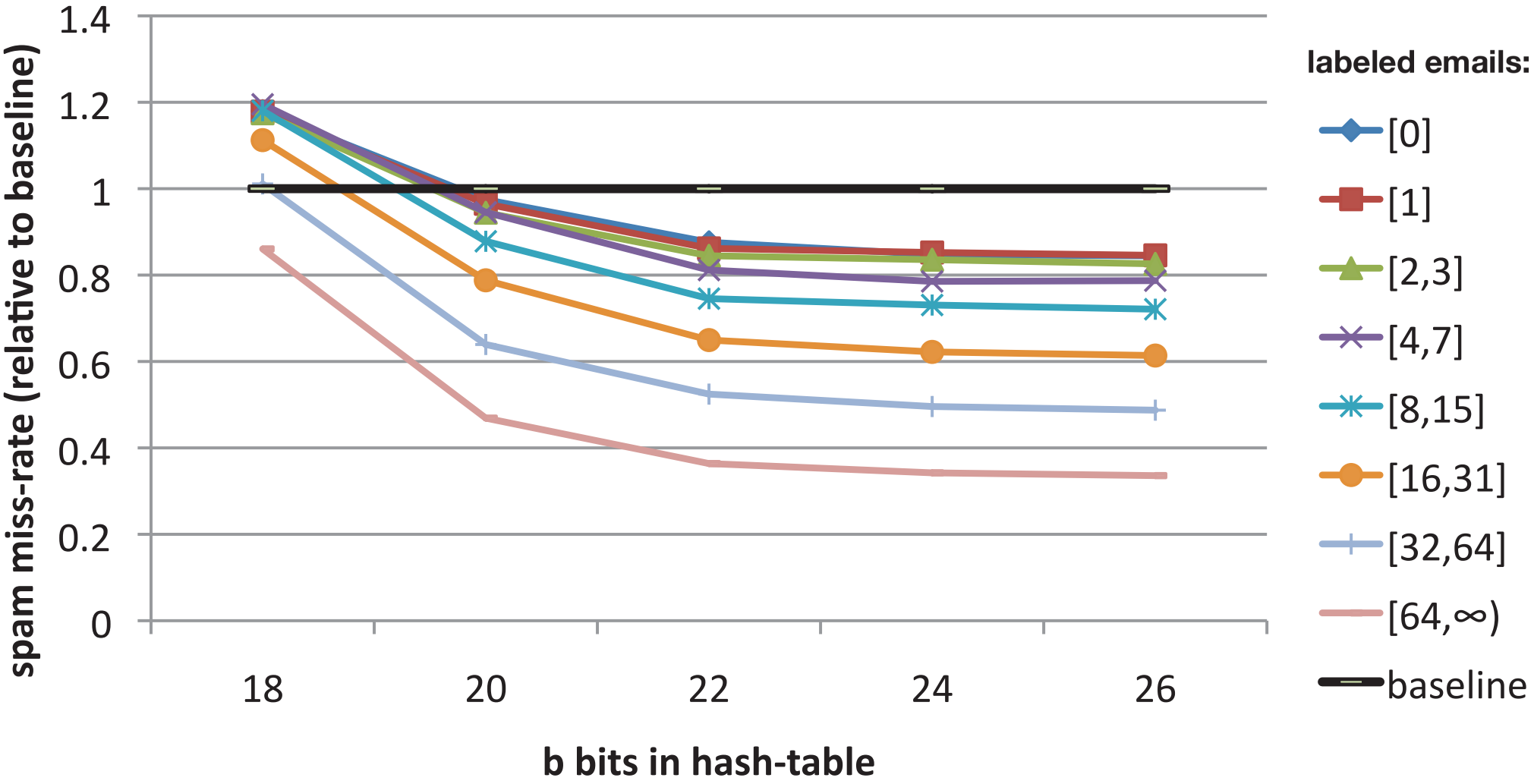


20 million emails, 400,000 users

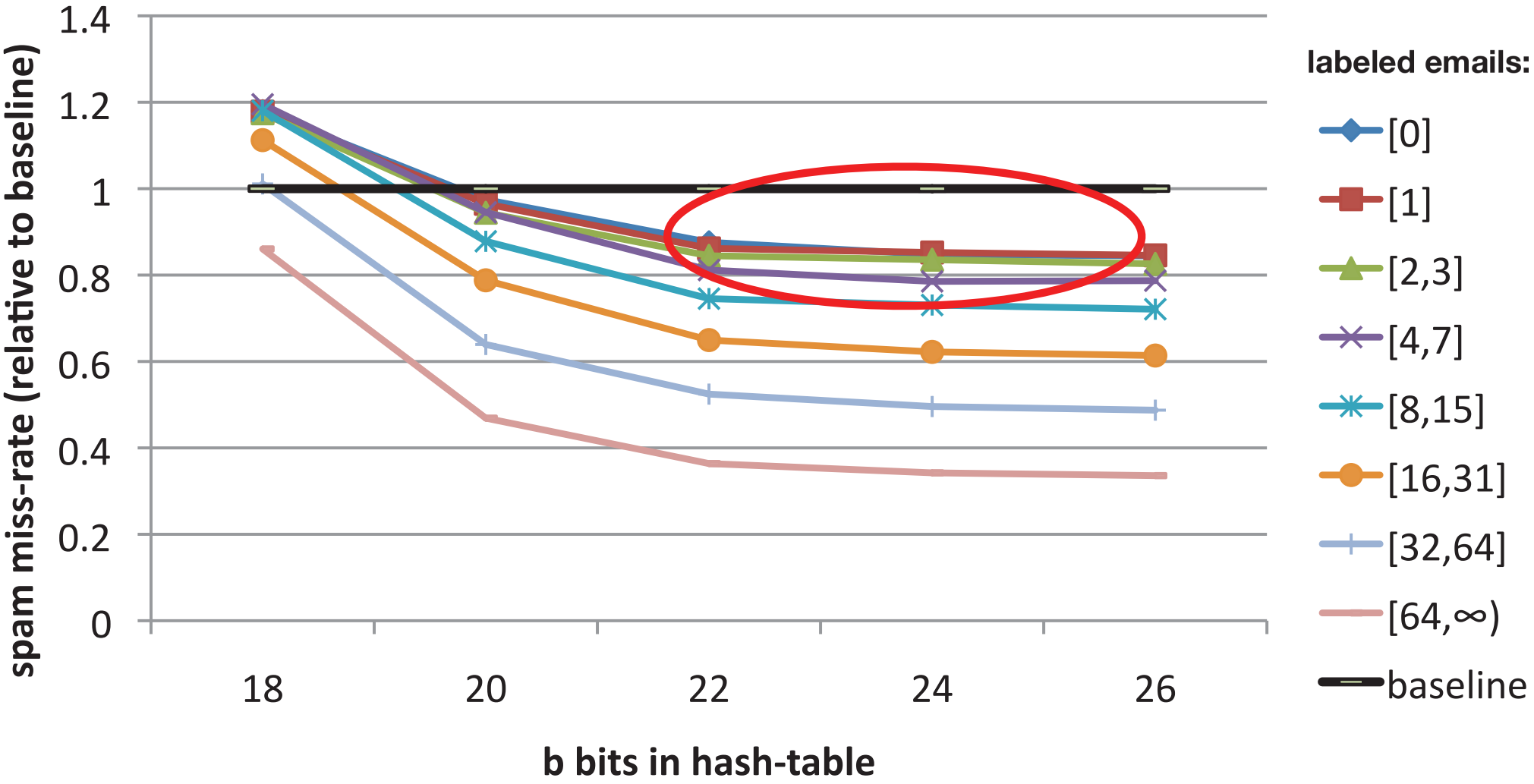
Lazy users …


Labeled emails per user
number of users
1000000
100000
10000
1000
100
10
1
0
13
26
39
52
65
78
91
104
117
130
143
156
169
182
197
211
228
244
261
288
317
370
523
number of labels

Results by user group



Results by user group



Approximate String Matches

- General idea

$$k(x, x') = \sum_{w \in x} \sum_{w' \in x'} \kappa(w, w') \text{ for } |w - w'| \leq \delta$$

Carnegie
Carneg1e
0arnegie
Canegie
Carn3gie

catch all
with wildcards

Carnegie
Carneg*e
*arnegie
Ca*negie
Carn*gie

- Map into fragments: `dog -> (*og, d*g, do*)`
- Hash fragments and weigh them based on mismatch amount
- Exponential in number of mismatches. Not alphabet size.

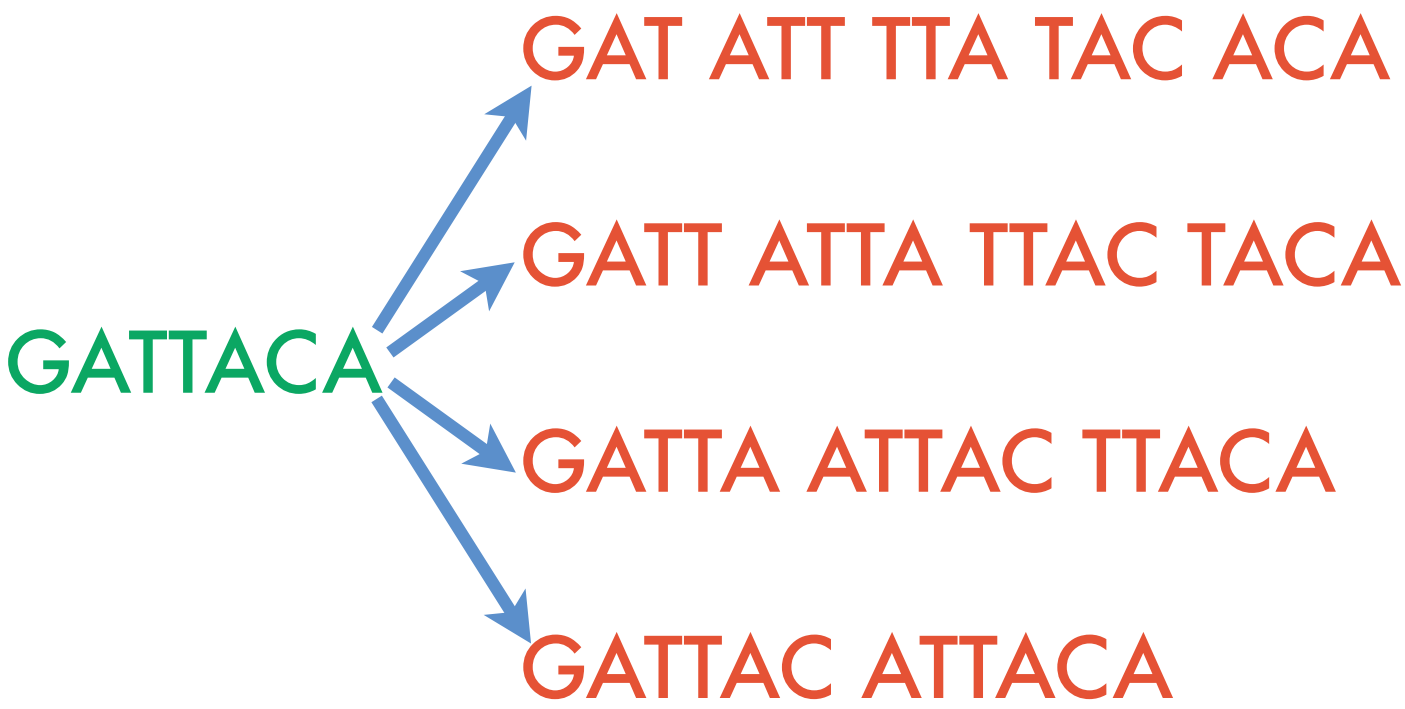
Fast String Kernels

- Example - DNA sequence

GATTACA

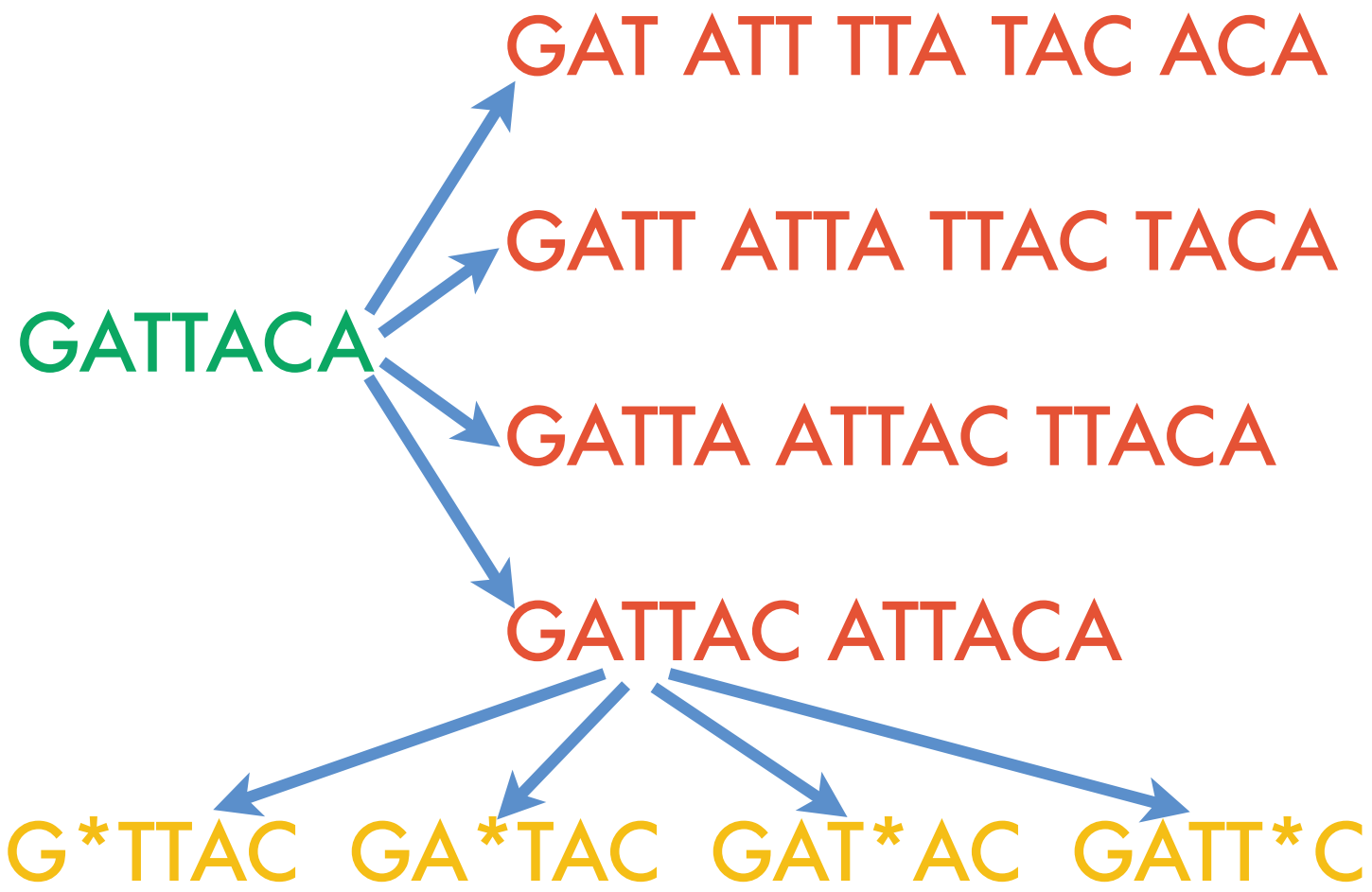
Fast String Kernels

- Example - DNA sequence



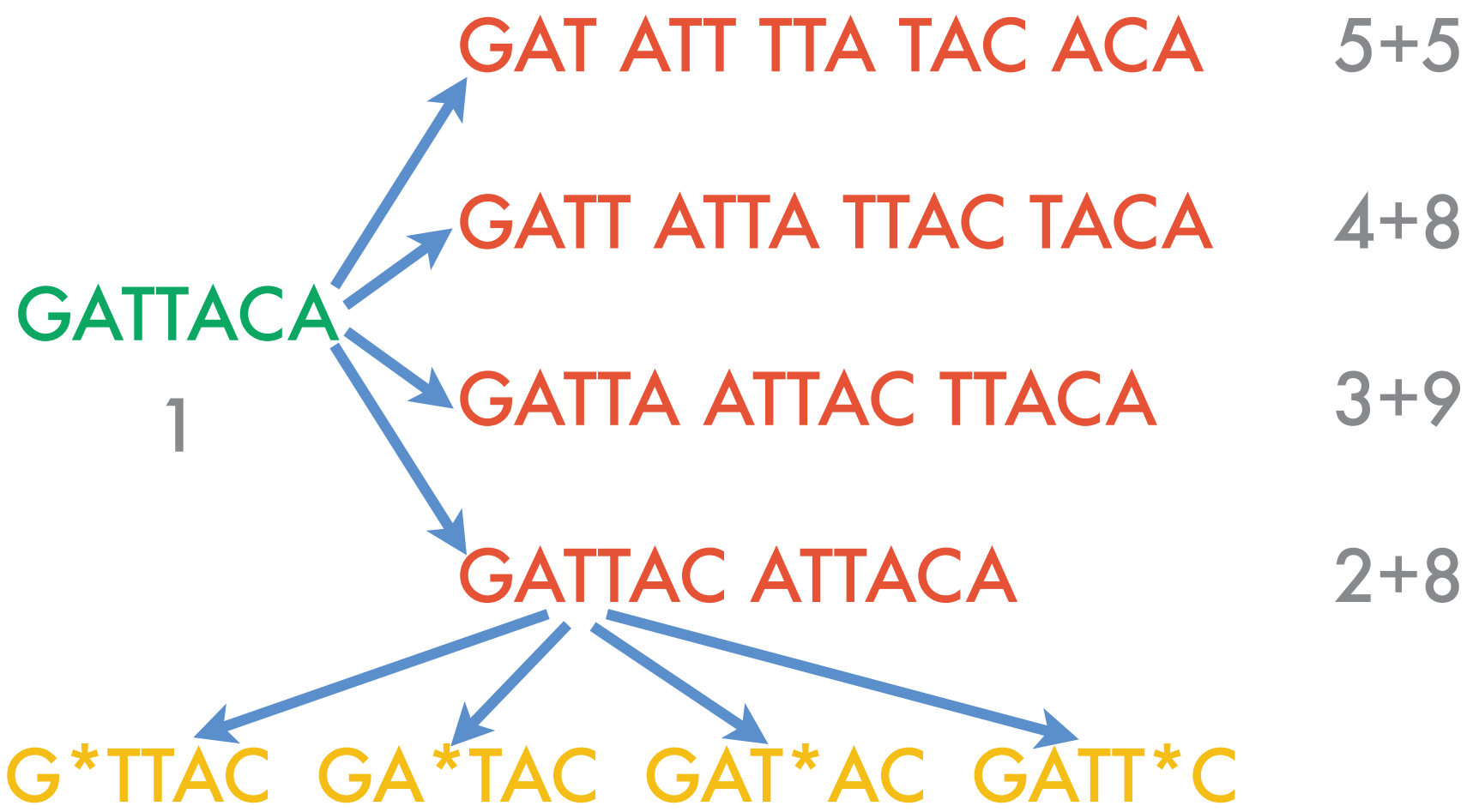
Fast String Kernels

- Example - DNA sequence



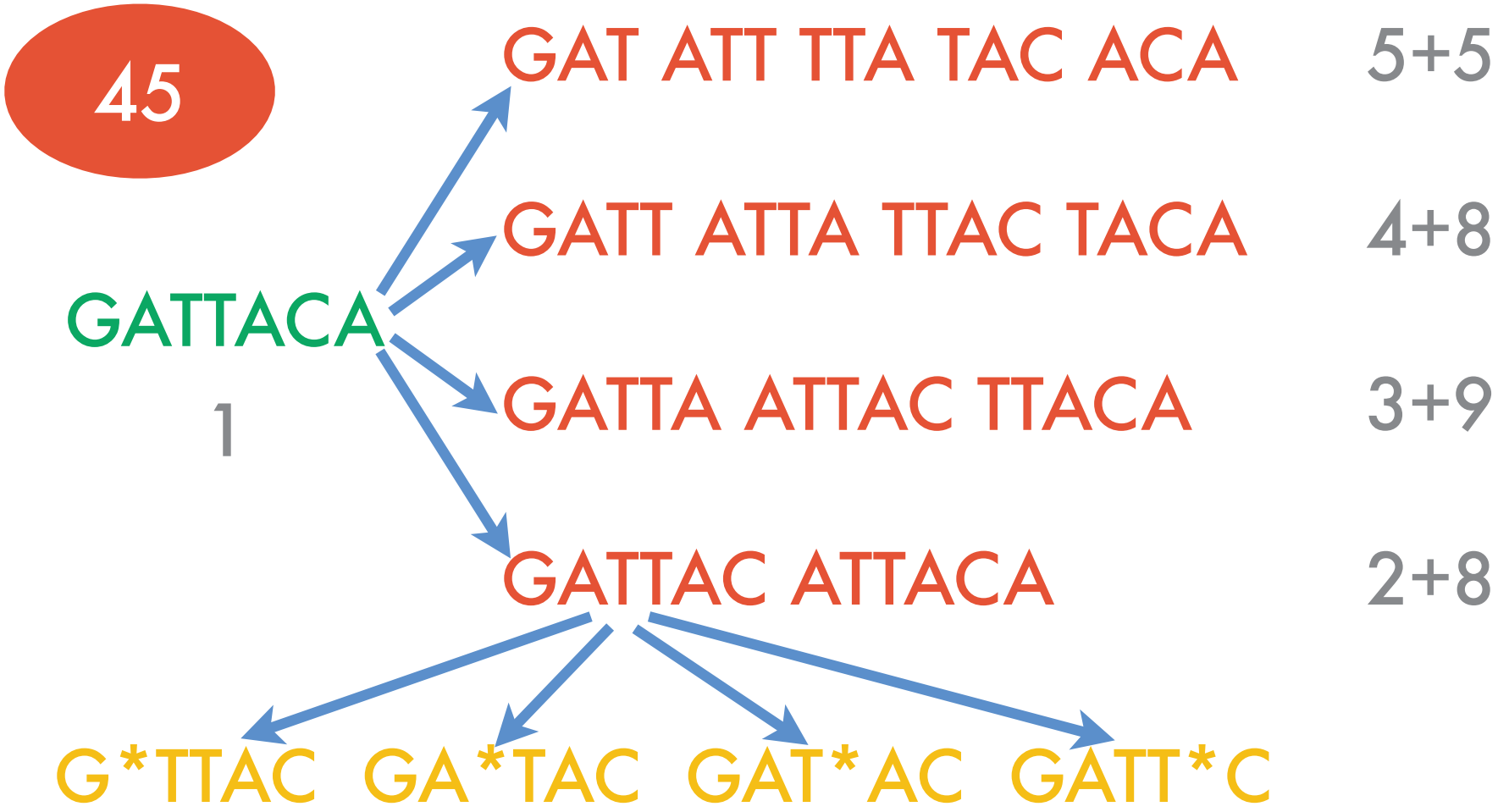
Fast String Kernels

- Example - DNA sequence



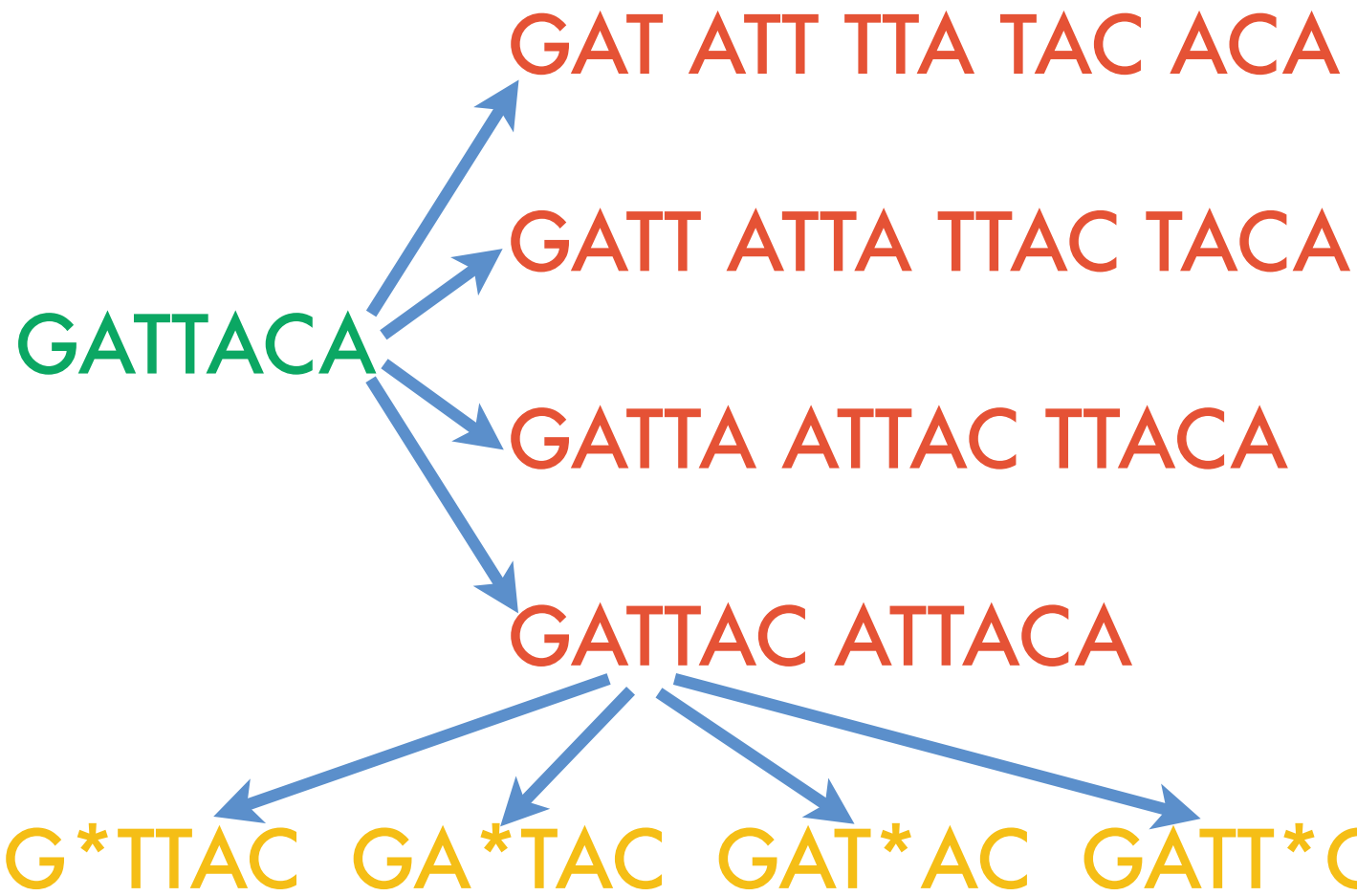
Fast String Kernels

- Example - DNA sequence



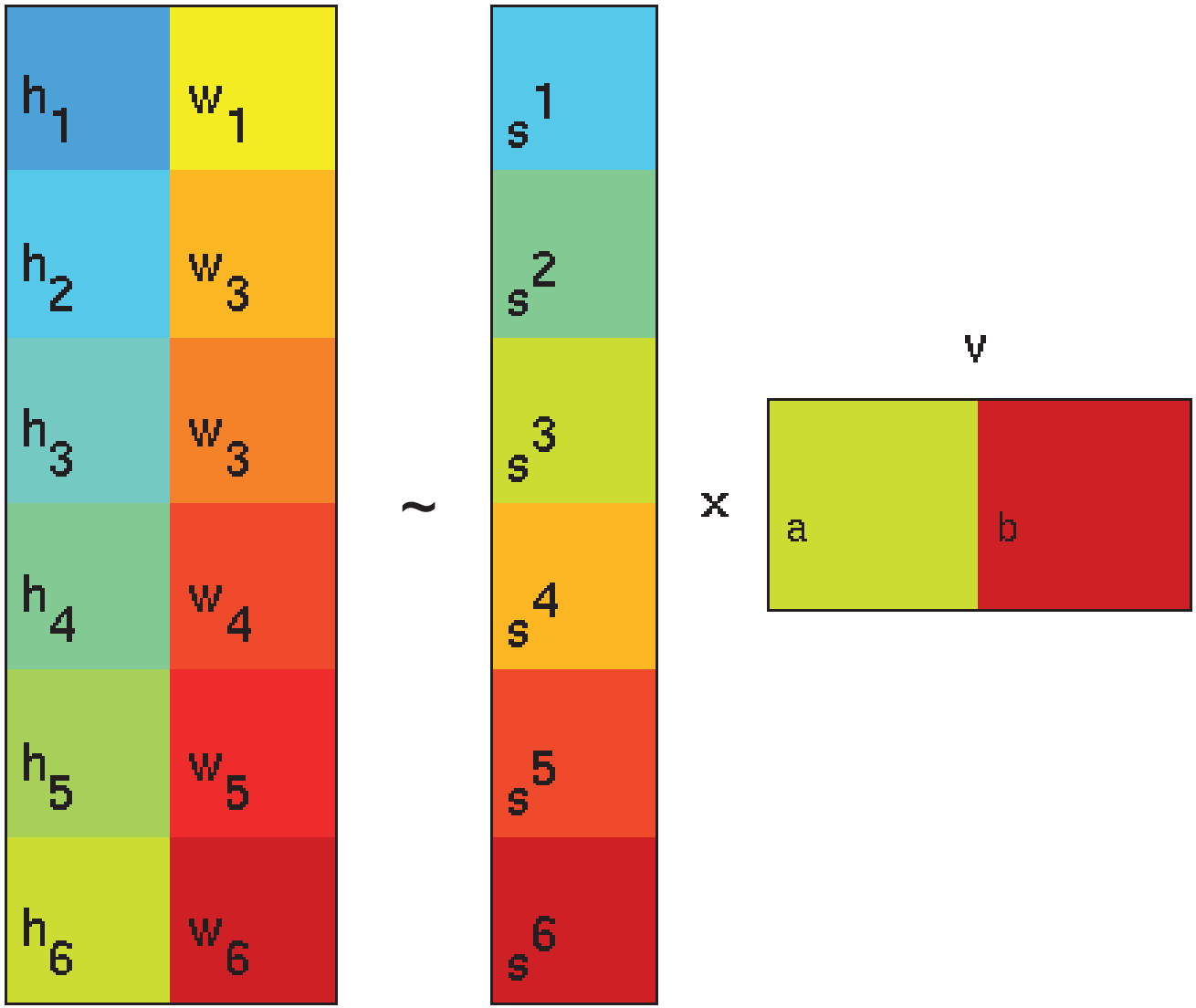
Fast String Kernels

- Store coefficients explicitly (complicated)
- Use hash kernel to update counts (trivial)



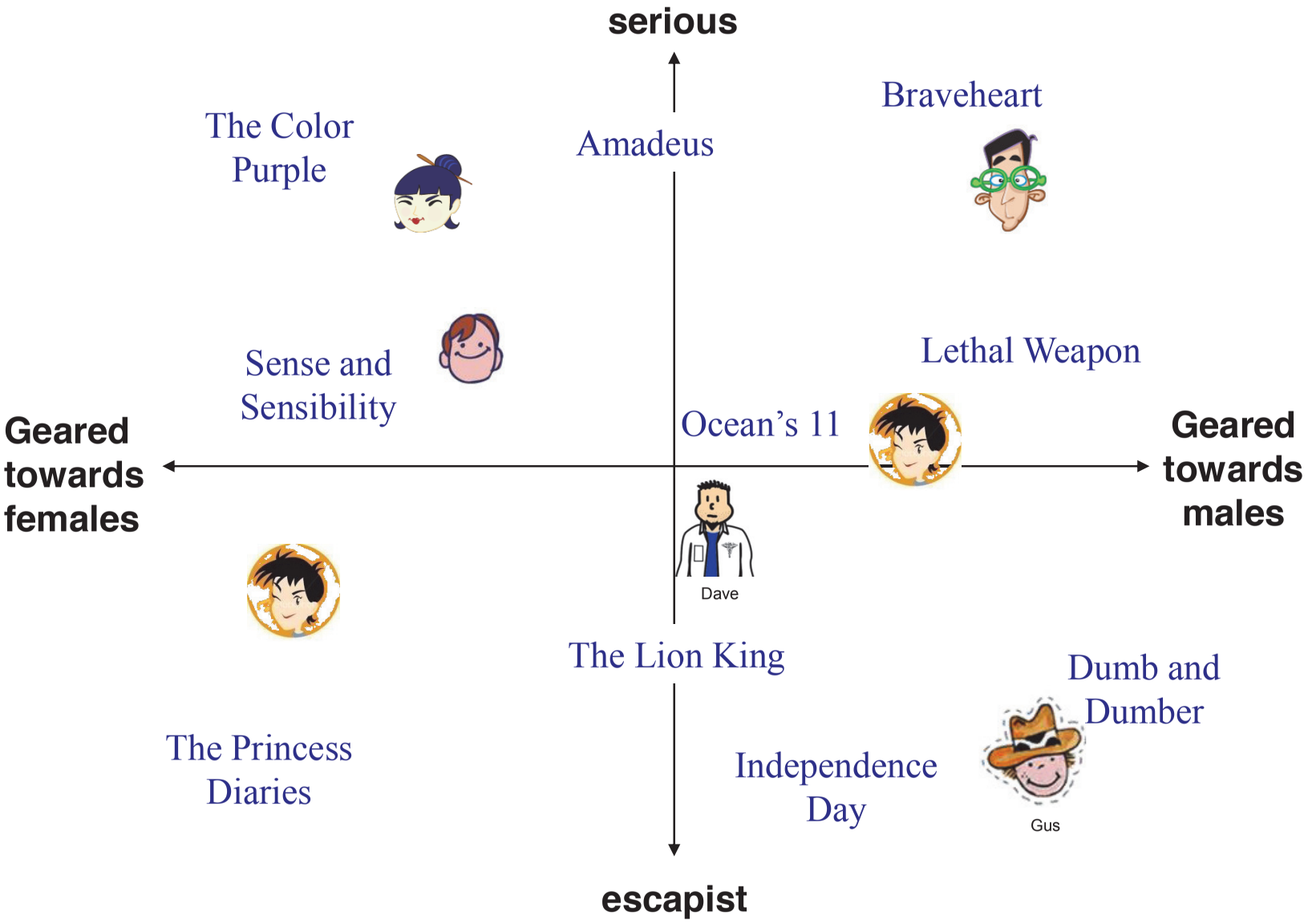
CoFi Rank & Matrix Compression

Basic Idea

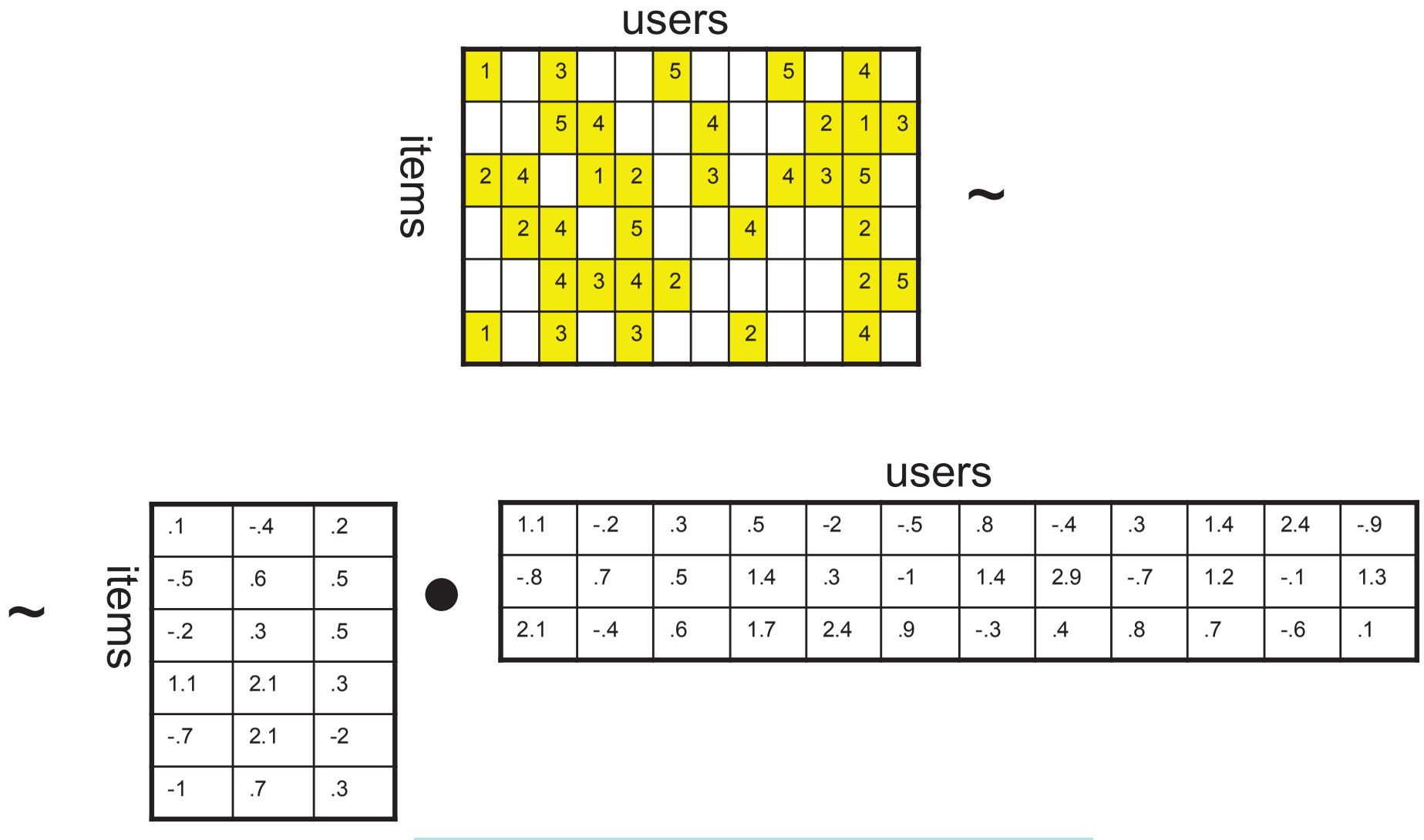


$$M \approx U \cdot V$$

Latent variable view



Basic matrix factorization



1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

Estimate unknown ratings as inner products of latent factors

users

items

1		3			5			5		4	
		5	?		4			2	1	3	
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

~

~

items

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

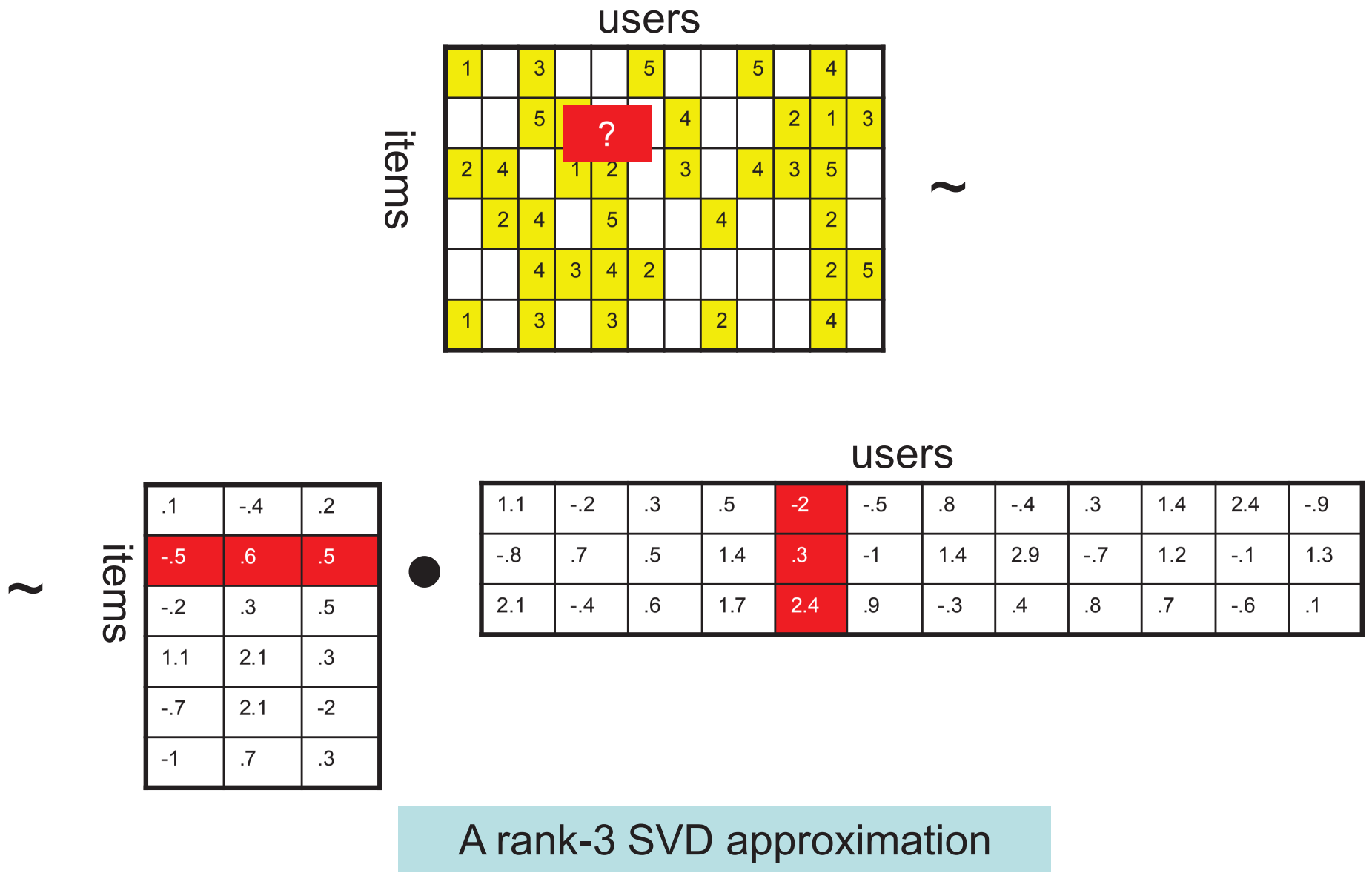
•

users

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

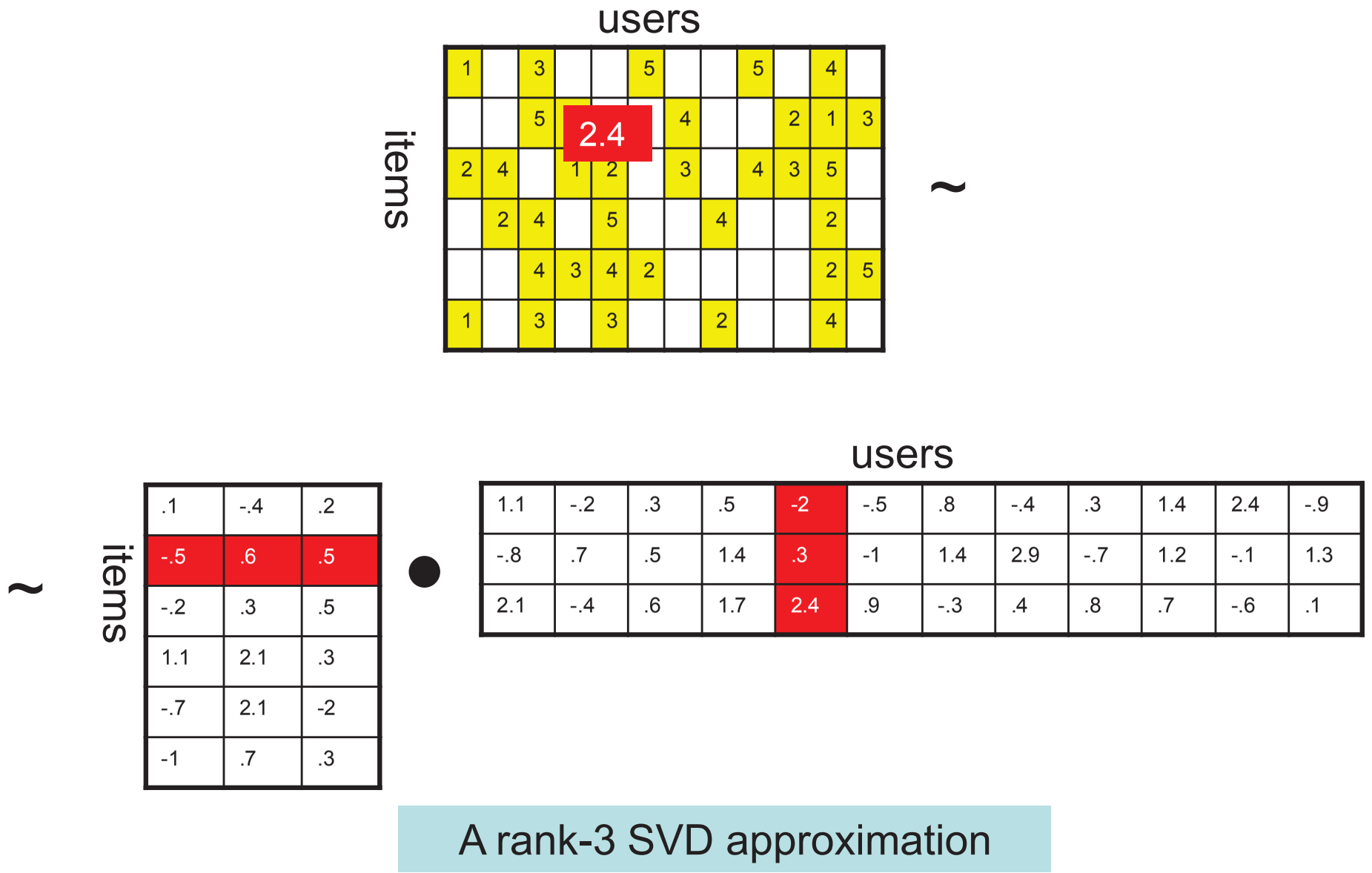
A rank-3 SVD approximation

Estimate unknown ratings as inner products of latent factors



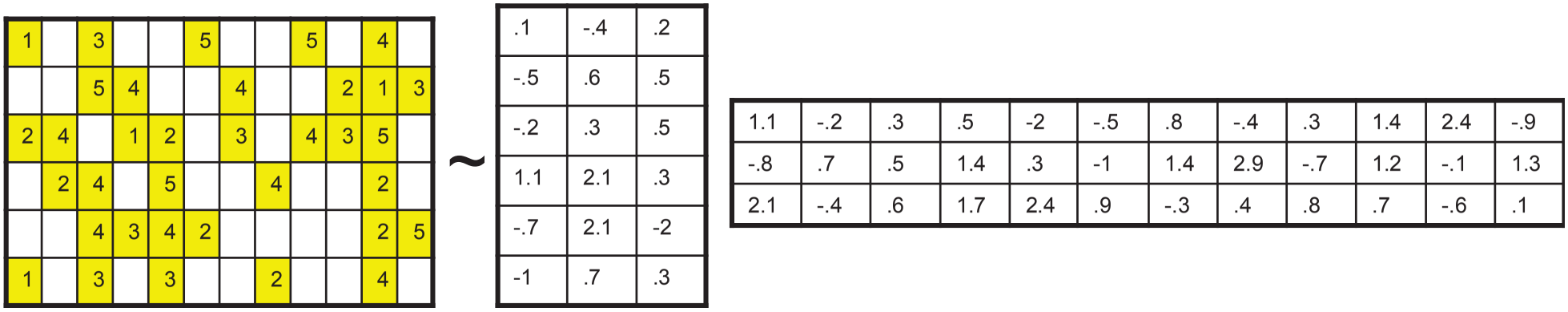
A rank-3 SVD approximation

Estimate unknown ratings as inner products of latent factors



A rank-3 SVD approximation

Properties



1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

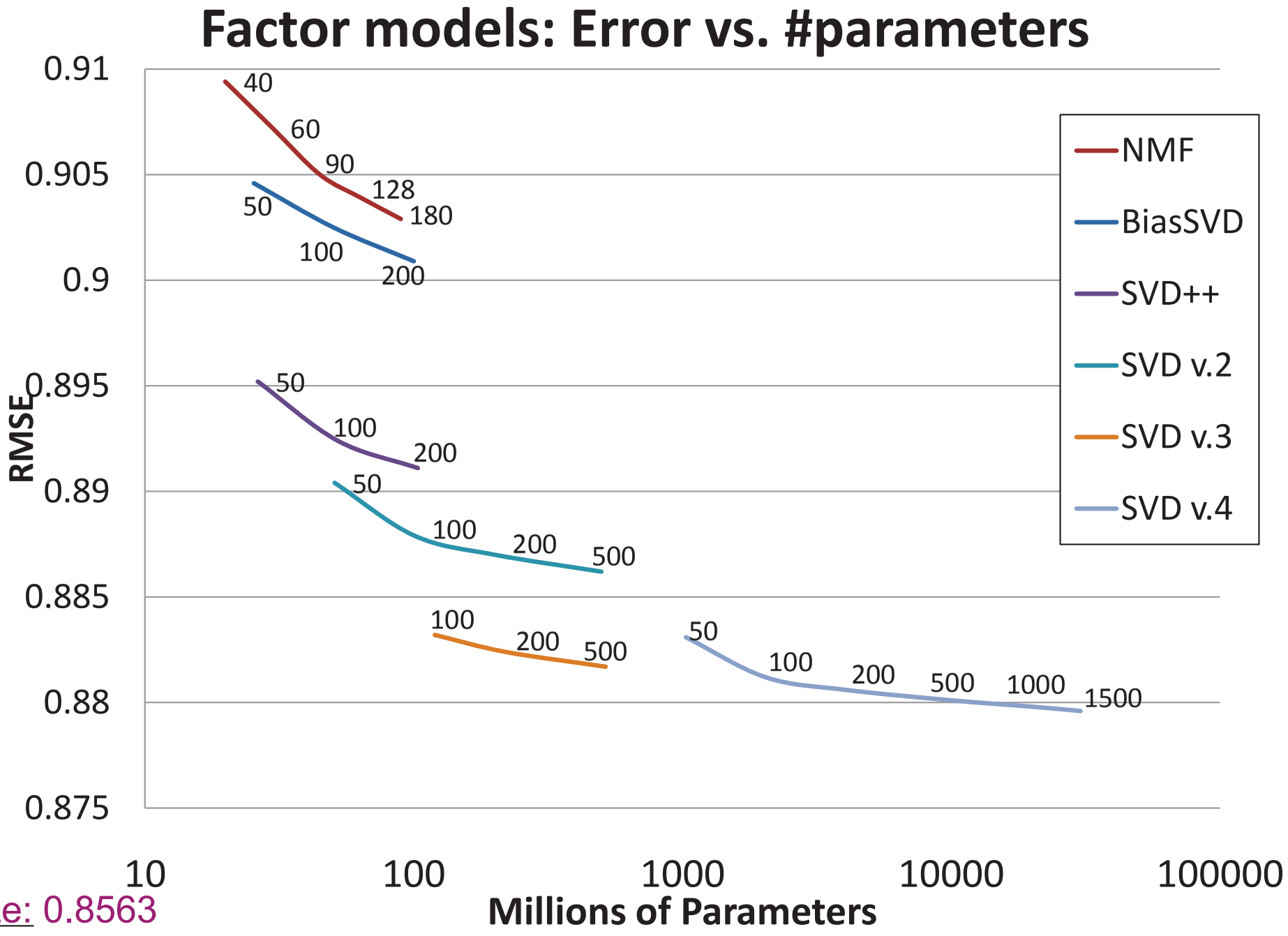
~

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

- SVD is undefined for missing entries
  - stochastic gradient descent (faster)
  - alternating optimization
- Overfitting without regularization particularly if fewer reviews than dimensions
- Very popular on Netflix

Netflix: 0.9514



Prize: 0.8563

Risk Minimization View

- **Objective Function**

$$\underset{p,q}{\text{minimize}} \sum_{(u,i)\in S} (r_{ui} - \langle p_u, q_i \rangle)^2 + \lambda \left[\|p\|_{\text{Frob}}^2 + \|q\|_{\text{Frob}}^2 \right]$$

- **Alternating least squares**

$$p_u \leftarrow \left[\lambda \mathbf{1} + \sum_{i|(u,i)\in S} q_i q_i^\top \right]^{-1} \sum_i q_i r_{ui}$$

$$q_i \leftarrow \left[\lambda \mathbf{1} + \sum_{u|(u,i)\in S} p_u p_u^\top \right]^{-1} \sum_i p_u r_{ui}$$

good for MapReduce

Risk Minimization View

- **Objective Function**

$$\underset{p,q}{\text{minimize}} \sum_{(u,i)\in S} (r_{ui} - \langle p_u, q_i \rangle)^2 + \lambda \left[\|p\|_{\text{Frob}}^2 + \|q\|_{\text{Frob}}^2 \right]$$

- **Stochastic gradient descent**

$$p_u \leftarrow (1 - \lambda\eta_t)p_u - \eta_t q_i (r_{ui} - \langle p_u, q_i \rangle)$$
$$q_i \leftarrow (1 - \lambda\eta_t)q_i - \eta_t p_u (r_{ui} - \langle p_u, q_i \rangle)$$

much faster

  - **No need for locking**
  - **Multicore updates asynchronously (Recht, Re, Wright, 2012 - Hogwild)**

Parameter Storage

- Similar problems as with linear models
  - Billions of users
  - Millions of products
- Storage - for 100 factors this requires
  $10^9 \times 8 = 800GB$
- We want a model that can be kept in RAM (<16GB)
  - Instant response for each user
  - Disks have 20 IOP/s at best (SSDs much better)
- Privacy (what if parameter vector leaks?)

Recall - Hash Kernel

instance:

Hey,

please mention subtly during your talk that people should use Yahoo mail more often.
Thanks,

Someone

task/user (=barney):

$$f(x) = \sum_i w[h(i)]\sigma(i)x_i$$

{-1, 1}

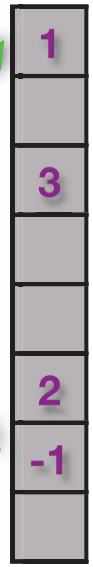
h('mention')

h('mention_barney')

$h()$

s(m_b)

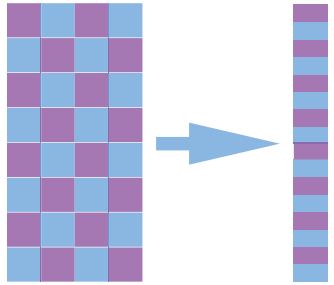
s(m)



Similar to count hash
(Charikar, Chen, Farrach-Colton, 2003)

Collaborative Filtering

- Hashing compression



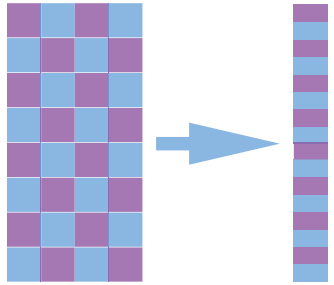
$$u_i = \sum_{j,k:h(j,k)=i} \xi(j,k)U_{jk} \text{ and } v_i = \sum_{j,k:h'(j,k)=i} \xi'(j,k)V_{jk}.$$

$$X_{ij} := \sum_k \xi(k,i)\xi'(k,j)u_{h(k,i)}v_{h'(k,j)}.$$

- Approximation is O(1/n)
  - To show that estimate is unbiased take expectation over Rademacher hash.

Parameter Storage

- Hashing compression



$$u_i = \sum_{j,k:h(k,j)=i} \xi(k,j)U_{kj} \text{ and } v_i = \sum_{j,k:h'(k,j)=i} \xi'(k,j)V_{kj}.$$

$$X_{ij} := \sum_k \xi(k,i)\xi'(k,j)u_{h(k,i)}v_{h'(k,j)}.$$

- Expectation

$$X_{ij} := \sum_k \xi(k,i)\xi'(k,j) \sum_{l,k:h(k,l)=h(k,i)} \sum_{o,k:h'(k,o)=h'(k,j)} \xi(k,l)\xi'(k,o)U_{kl}V_{ko}$$

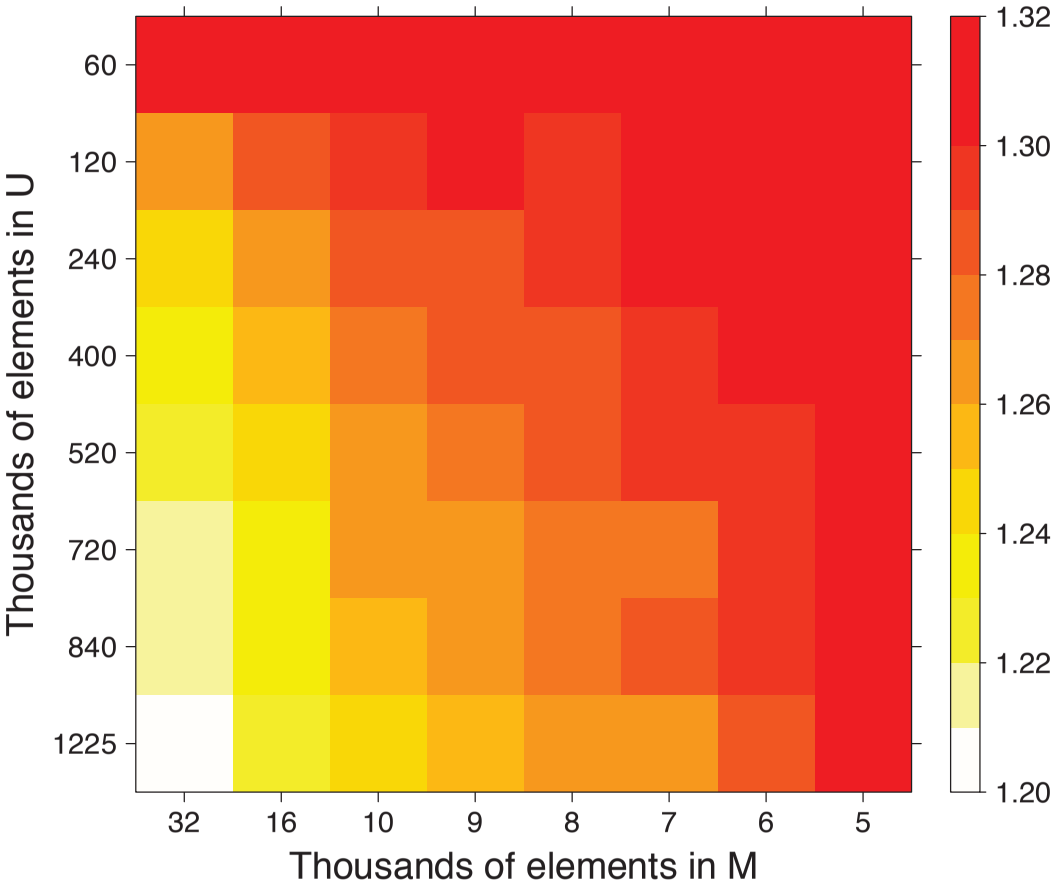
expectation vanishes

Collaborative Hashing

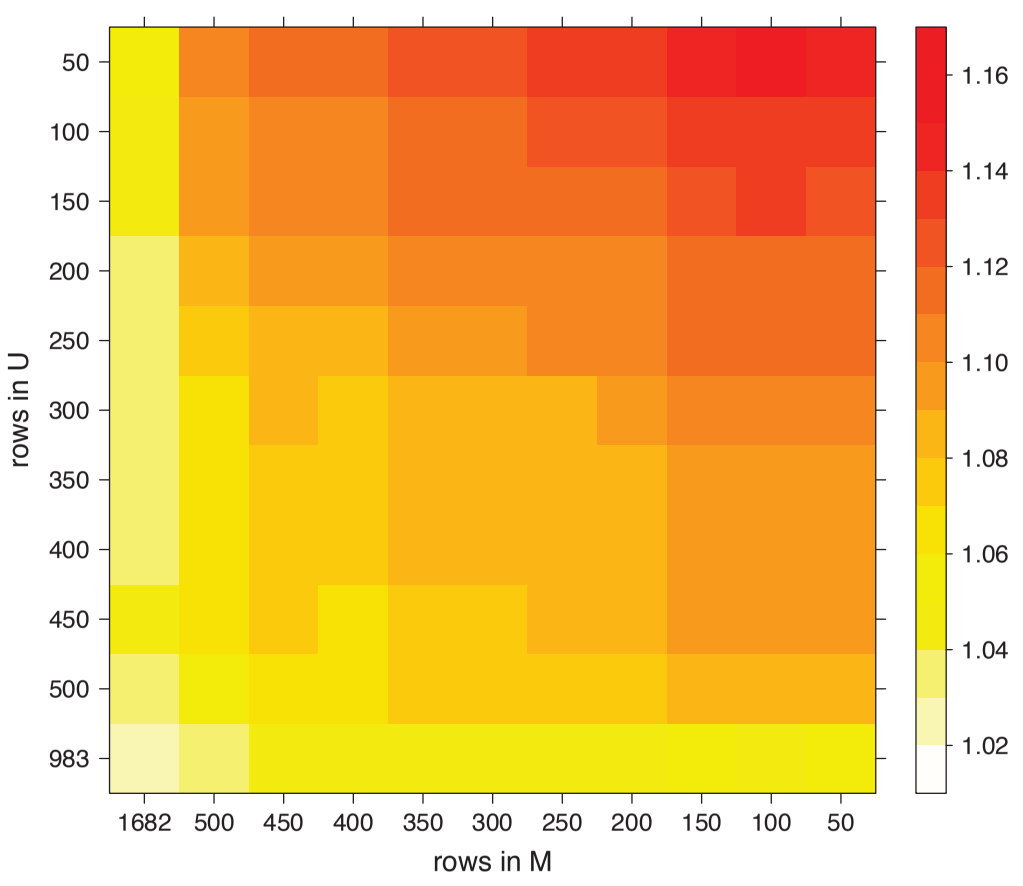
- Combine with stochastic gradient descent
- Random access in memory is expensive (we now have to do k lookups per pair)
- Feistel networks can accelerate this
- Distributed optimization without locking (Hogwild, Async-ADMM, Distributed Prox)

- Collaborative Filtering on a Budget , AISTATS 2010, Sardinia, Italy Mai 13-15
- Maximum Margin Code Recommendation, ACM Recommender Systems (RecSys), New York, USA, October 22-25, 2009
- Adaptive collaborative filtering, ACM Recommender Systems (RecSys), Lausanne, Switzerland, October 23-25, 2008
- Improving maximum margin matrix factorization, Machine Learning Journal and European Conference on Machine Learning (ECML/PKDD)
- CofiRank - Maximum Margin Matrix Factorization for Collaborative Ranking, Neural Information Processing Systems (NIPS) 2007

Examples



Eachmovie



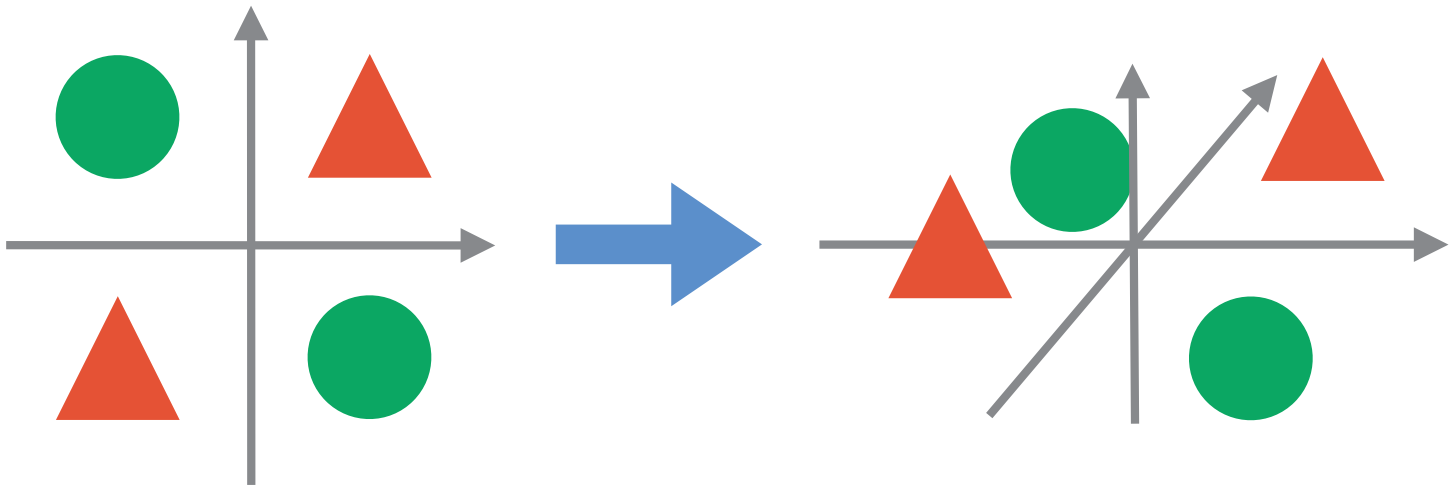
MovieLens

- Sparse Aggregators
  (Bloom, CountMin)
- Optimizing over Sparse Aggregators
  (Linear models, CoFi rank)
- **Random function classes**
  (Random Kitchen Sinks, Fast Food)
- Random Projections
  (LSH, SimHash, FastEx)


Kernels

The beauty of kernels

- Classical algorithms using inner products (classification, regression, ICA, PCA, distribution tests, exponential families, demand elasticity ...)
- Make nonlinear by mapping into feature space

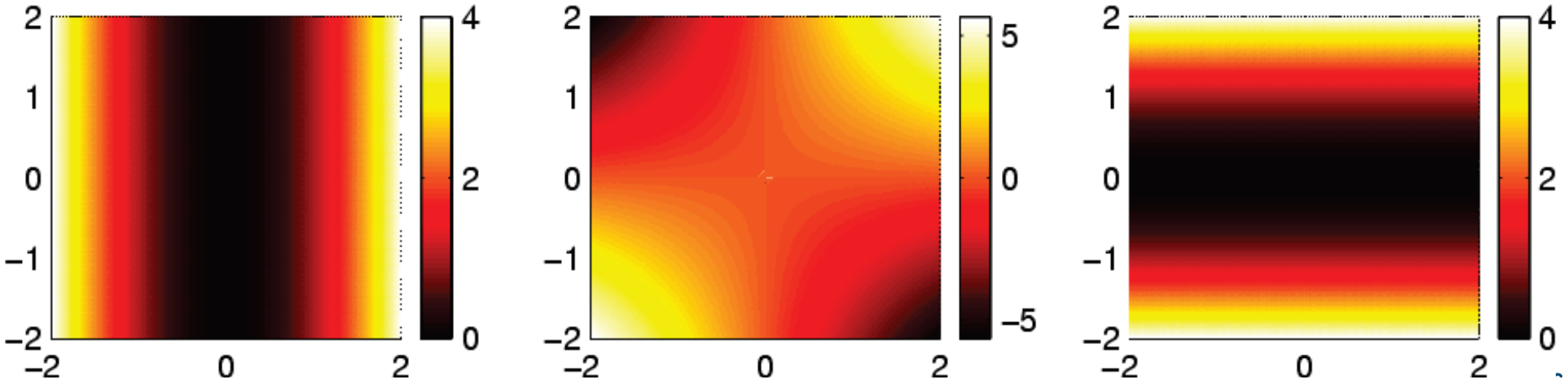


(x_1, x_2) $\quad x \to \phi(x)$ $\quad (x_1, x_2, x_1 x_2)$

- Avoid computational cost via $k(x, x') = \langle \phi(x), \phi(x') \rangle$

The beauty of kernels

- Classical algorithms using inner products (classification, regression, ICA, PCA, distribution tests, exponential families, demand elasticity ...)
- Make nonlinear by mapping into feature space



- Avoid computational cost via $k(x, x') = \langle \phi(x), \phi(x') \rangle$

The trouble with kernels

- Representer theorem $f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x)$
- Number of basis functions increases linearly with sample size

training

testing

naive approach

The trouble with kernels

- Representer theorem $f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x)$
- Number of basis functions increases linearly with sample size

Burges 1996, Schölkopf et al. 1998, Romdhani et al. 2000

The trouble with kernels

- Representer theorem $$f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x)$$
- Number of basis functions increases linearly with sample size (Steinwart and Christmann, 2008)

Williams & Seeger, 2001, Smola & Schölkopf, 2001
Scheinberg & Fine 2004, Kempe & Das 2010

The trouble with kernels

$$f(x) = \sum_{i=1}^{n} w_i \phi_i(x)$$

- Back to primal space
- Number of basis functions (should) increase with sample size
- Explicit formulation not always possible (and kernels in infinite dimensions)

training

testing

random kitchen sinks

Neal 1994, Rahimi & Recht 2008, 2009

The trouble with kernels

- Kernel expansion $f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x)$
- Function expansion $f(x) = \sum_{i=1}^{n} w_i \phi_i(x)$
- Slow approximations (d dimensions, n features, m samples)

	CPU Training	CPU Test	RAM Training	RAM Test
Naive	$O(m^2 d)$	$O(md)$	$O(md)$	$O(md)$
Reduced set	$O(m^2 d)$	$O(nd)$	$O(md)$	$O(nd)$
Low rank	$O(mnd)$	$O(nd)$	$O(nd)$	$O(nd)$
Random Kitchen Sinks	$O(mnd)$	$O(nd)$	$O(nd)$	$O(nd)$


Fast Food

Fourier Representation of Kernels

- Kernel invariant under group action

$$k(x, x') = k(\rho \circ x, \rho \circ x') \text{ for all } \rho \in G$$

- Expansion in irreducible representations of symmetry group

$$k(x, x') = \int \lambda_z \phi_z(x) \phi_z(x') \text{ where } \phi_z(\rho \circ x) = U_\rho \phi_{r(z)}(x)$$

- Radial basis function kernels in Fourier expansion (Bochner 1932)

$$k(x, x') = k(\|x - x'\|) = \int d\omega \kappa(\omega) e^{i\langle \omega, x \rangle} e^{-i\langle \omega, x' \rangle}$$

- Inner product kernels in spherical harmonics (Schoenberg 1934)

$$k(x, x') = k(\langle x, x' \rangle) = \sum_{jlm} c_{jlm} Y_{jl}\left(\frac{x}{\|x\|}\right) \bar{Y}_{jl}\left(\frac{x'}{\|x'\|}\right) (\|x\| \, \|x'\|)^m$$

- Symmetric group analogous

Random Kitchen Sinks

(Rahimi & Recht 2008, 2009)

- Radial basis function kernels in Fourier expansion

$$k(x, x') = k(\|x - x'\|) = \int d\omega \kappa(\omega) e^{i\langle \omega, x \rangle} e^{-i\langle \omega, x' \rangle}$$

- Draw frequencies from p(w) and approximate kernel

$$k(x, x') = \frac{1}{n} \sum_{j=1}^{n} e^{i\langle \omega_j, x \rangle} e^{-i\langle \omega_j, x' \rangle} \text{ where } \omega_j \sim \kappa(\omega)$$

  - For Gaussian RBFs draw w_j from a Gaussian
  - In general, draw w from spherically symmetric distribution (and rescale suitably)
- Dominant cost (store all w_j and multiply)

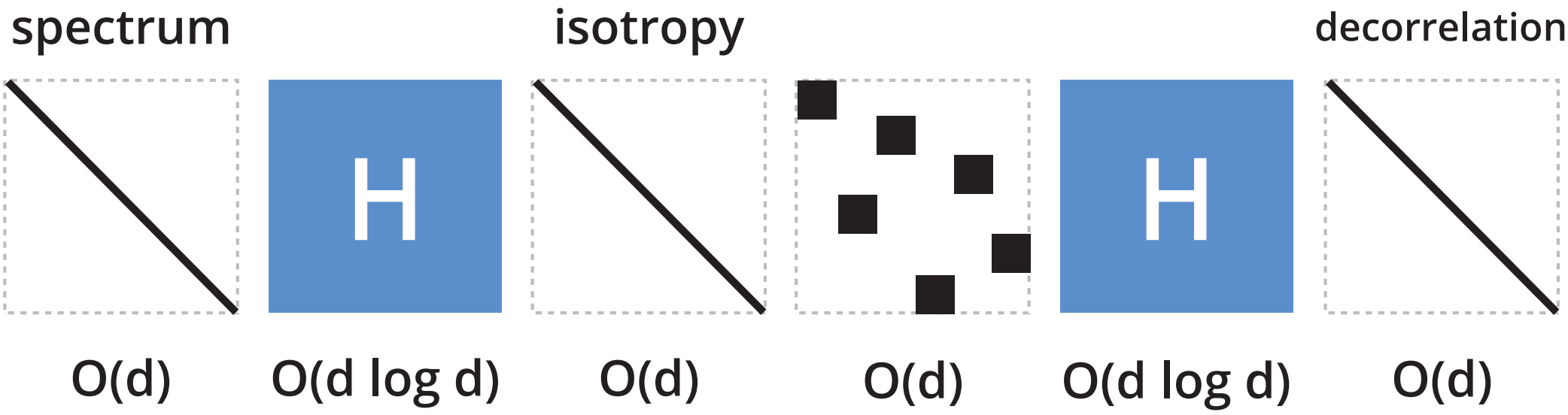
$$\Omega x \text{ is } O(nd) \text{ CPU and } O(nd) \text{ RAM}$$

- Accelerate this. Smaller memory footprint

Fastfood

- Gaussian matrix Ω costs O(nd) per multiplication
  - O(nd) for storage on a random matrix
  - O(nd) computation for multiplying by a random matrix
- Fake Gaussian (for the moment assume square matrices)

$$\tilde{M} = SHG\Pi HB$$



- Fast & cheap: multiplication is O(d log d), storage is O(d).

Fastfood

- Gaussian matrix Ω costs O(nd) per multiplication
  - O(nd) for storage on a random matrix
  - O(nd) computation for multiplying by a random matrix
- Fake Gaussian (for the moment assume square matrices)

$$\tilde{M} = SHG\Pi HB$$

  - S is random diagonal scaling matrix (deals with spectrum)
  - H is Hadamard matrix admitting O(d log d) multiply

$$H_{2n} = \begin{bmatrix} H_n & H_n \\ H_n & -H_n \end{bmatrix} \text{ and } H_1 = 1$$

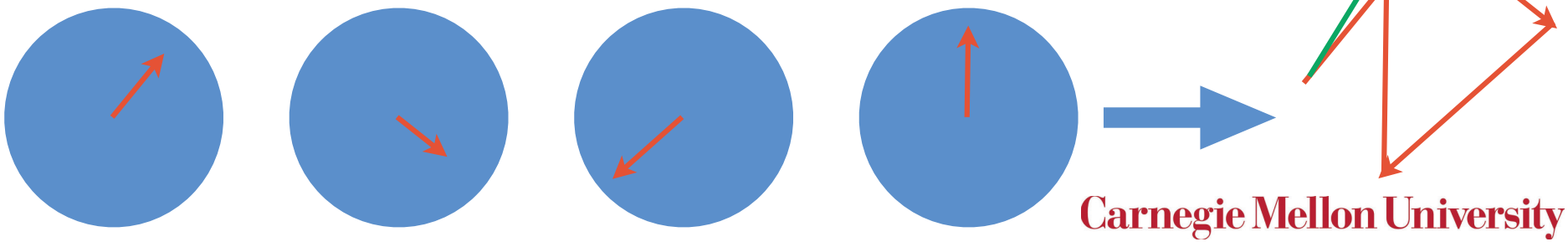
  - G is random diagonal Gaussian matrix
  - Π is random permutation matrix
  - B is random binary {-1, 1} diagonal matrix
- Fast & cheap: multiplication is O(d log d), storage is O(d).

Fastfood - is it safe?

- It's fast but is it good? $\tilde{M} = SHG\Pi HB$
- Theorem - In expectation the kernel is correct.
- Proof - Each row of M (on its own) is Gaussian.
- Corollary - Other kernels by different rescaling S
- Example - Matern kernel

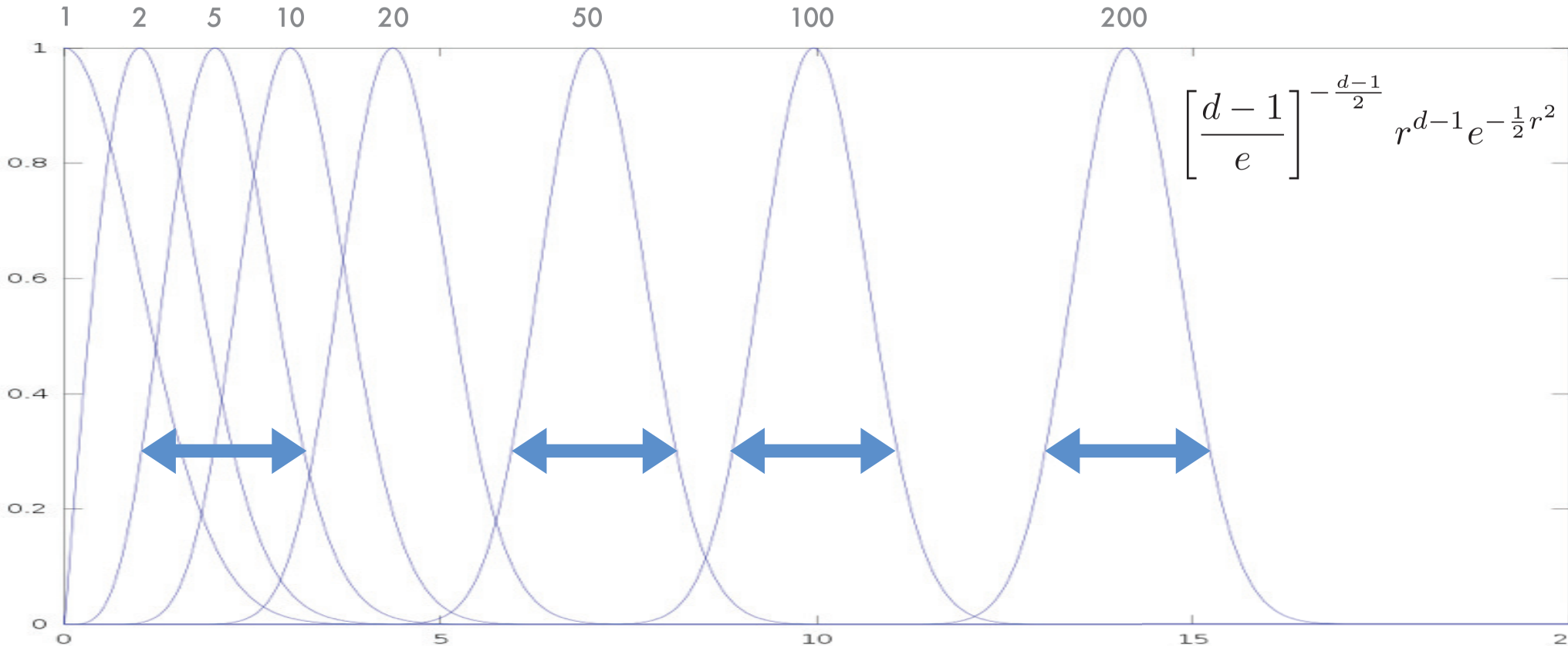
$$k(x, x') = \|x - x'\|^{-\frac{nd}{2}} J^n_{\frac{d}{2}}(\|x - x'\|)$$

Fourier transform is n-fold convolution of unit ball.
(to sample S draw n independent points and add)



Why Gauss is bad for you

- Concentration of measure for draws from Gaussian
- Mode in d dimensions at $\sqrt{d-1}$ for $r^{d-1}e^{-\frac{1}{2}r^2}$
- Explains why Gaussian RBF not great for large d



Fastfood - is it really safe?

- **Bounds on the variance (v = x-x')**

$$\mathrm{Var}\left[\psi_j(v)\right] = \frac{1}{2}\left(1 - e^{-\|v\|^2}\right)^2$$

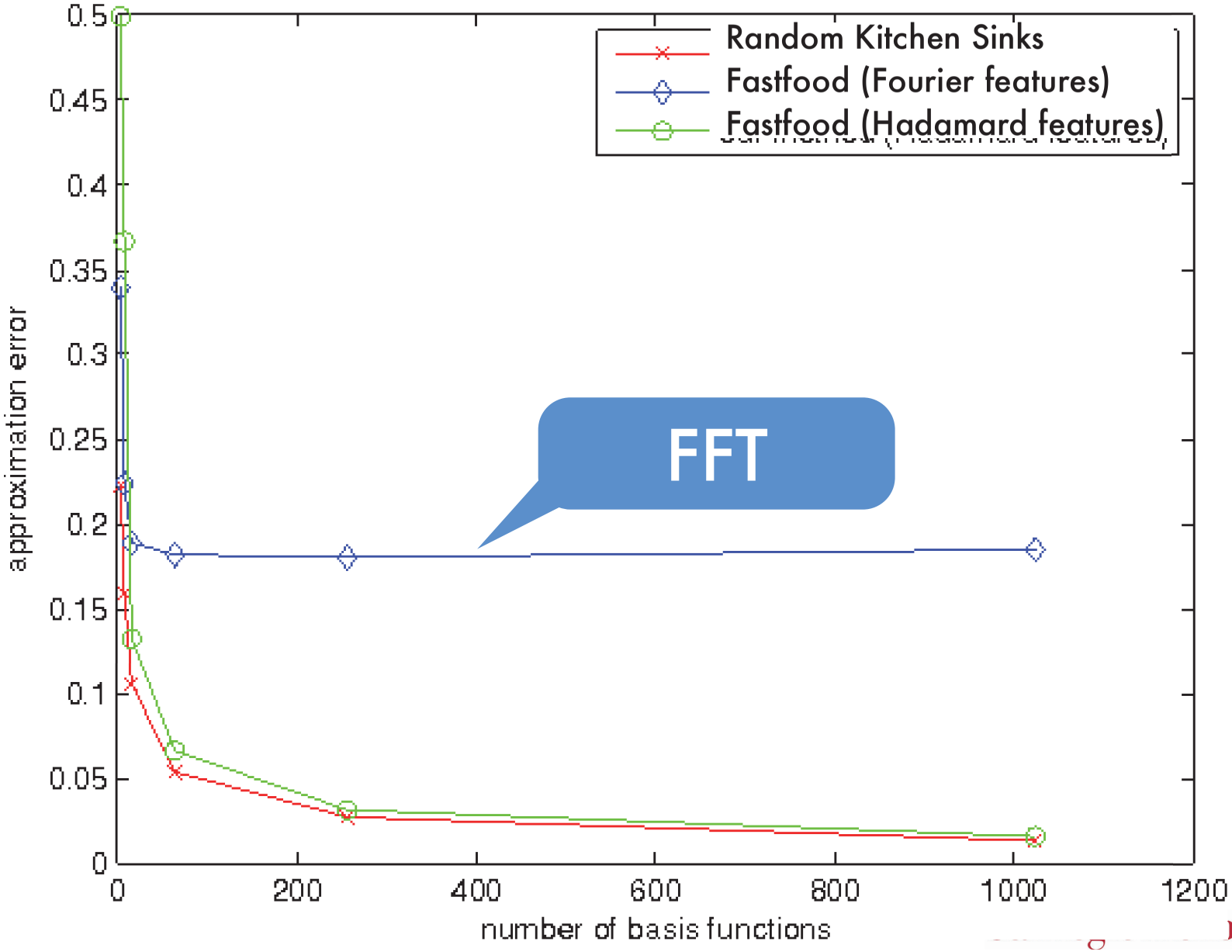
$$\mathrm{Var}\left[\sum_{j=1}^{d} \psi_j(v)\right] \leq \frac{d}{2}\left(1 - e^{-\|v\|^2}\right)^2 + dC(\|v\|)$$

$$where\ C(\alpha) = 6\alpha^4\left[e^{-\alpha^2} + \frac{\alpha^2}{3}\right].$$

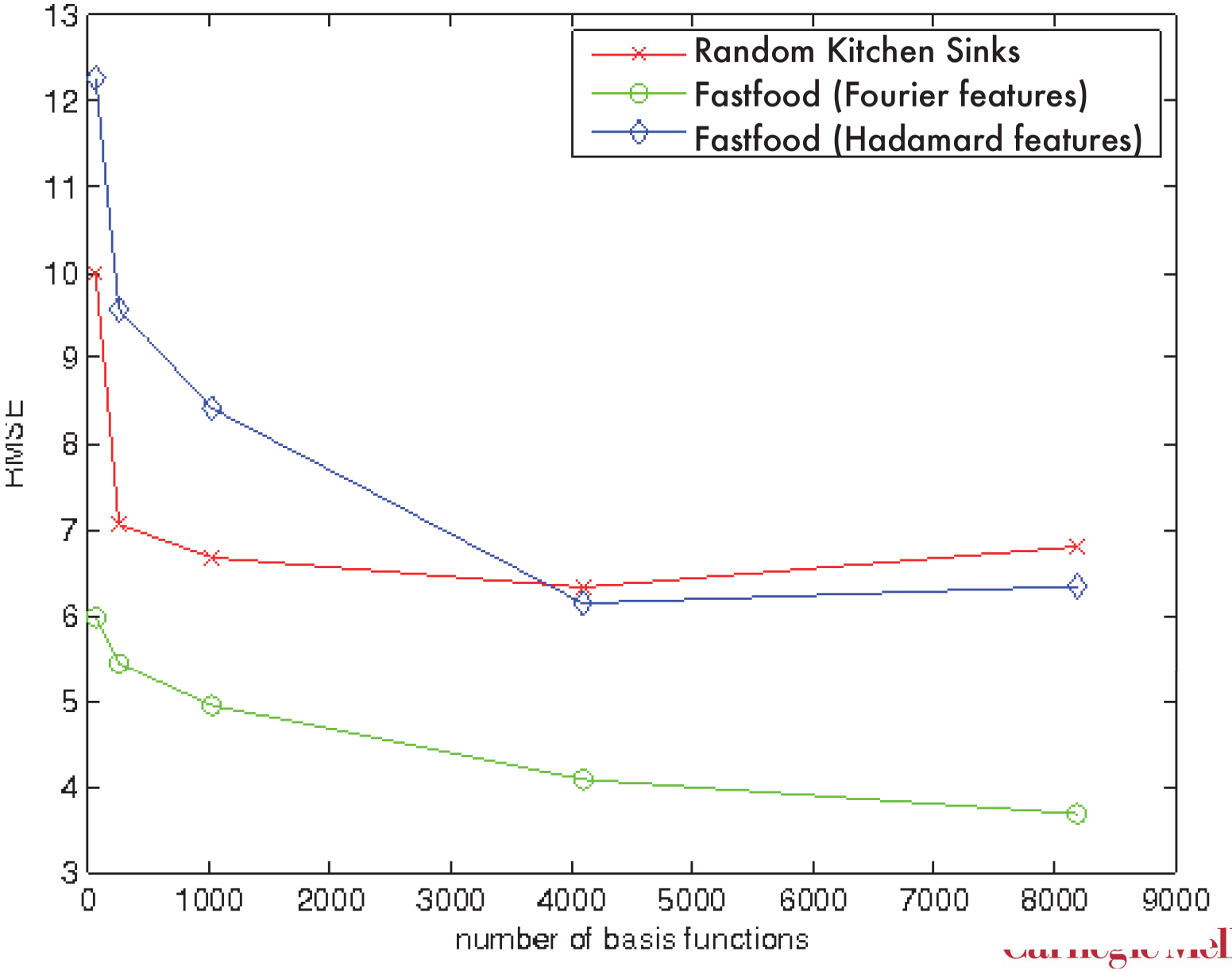
- **Things are very well behaved as long as features are properly rescaled**
- **Proof by brute force integration (similar to Dasgupta et al., 2012)**


The proof is in the pudding

Matrix approximation error



Generalization Performance (UCI CPU dataset)



Speed & accuracy

Dataset	m	d	Exact	Nystrom	Random Kitchen Sinks	Fastfood FFT	Fastfood
Insurance Company (COIL2000)	5,822	85	0.231	0.232	0.266	0.266	0.264
Wine Quality	4,080	11	0.819	0.797	0.740	0.721	0.740
Parkinson Telemonitor	4,700	21	0.059	0.058	0.054	0.052	0.054
CPU	6,554	21	7.271	6.758	7.103	4.544	7.366
Relative location of CT slices (axial)	42,800	384	n.a.	60.683	49.491	58.425	43.858
KEGG Metabolic Reaction Network	51,686	27	n.a.	17.872	17.837	17.826	17.818
Year Prediction MSD	463,715	90	n.a.	0.113	0.123	0.106	0.115
Forest	522,910	54	n.a.	0.837	0.840	0.838	0.840

faster

works fine

d	n	Fastfood	RKS	Speedup	RAM
1,024	16,384	0.00058s	0.0139s	24x	256x
4,096	32,768	0.00136s	0.1224s	90x	1024x
8,192	65,536	0.00268s	0.5360s	200x	2048x

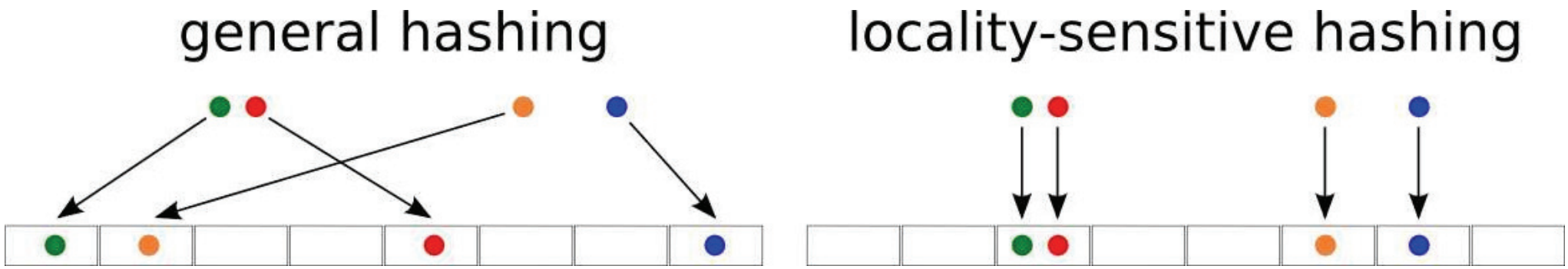
State of the art on CIFAR-10 (63%)
for permutation invariant classifiers

Gaussians considered harmful

Random Kitchen Sinks	Fastfood FFT	Fastfood	Fastfood Matern
0.266	0.266	0.264	0.235
0.740	0.721	0.740	0.720
0.054	0.052	0.054	0.052
7.103	4.544	7.366	4.211
49.491	58.425	43.858	14.868
17.837	17.826	17.818	17.846
0.123	0.106	0.115	0.116
0.840	0.838	0.840	0.976

- Sparse Aggregators
  (Bloom, CountMin)
- Optimizing over Sparse Aggregators
  (Linear models, CoFi rank)
- Random function classes
  (Random Kitchen Sinks, Fast Food)
- **Random Projections**
  (LSH, SimHash, FastEx)

Locality Sensitive Hashing



Locality Sensitive Hashing

- **Basic idea**
  Dimensionality reduction by multiplying with a dense random matrix (e.g. Gaussian, stable …)

$$z = Px$$

- **Preserves distances**
  For any set of n points and $k \geq 4(\epsilon^2/2 - \epsilon^3/3)^{-1} \log n$
  we can find P (e.g. Gaussian and normalize) such that

$$(1-\epsilon)\left\|Px_i - Px_j\right\|^2 \leq \left\|x_i - x_j\right\|^2 \leq (1+\epsilon)\left\|Px_i - Px_j\right\|^2$$

  - Dimensionality independent
  - Lots of papers by Indyk, Achlioptas, Immorlica, Motwani, Dasgupta …

Locality Sensitive Hashing

- Basic idea
  Dimensionality reduction by multiplying with a dense random matrix (e.g. Gaussian, stable ...)

$$z = Px$$

- Document classification $\langle w, x \rangle \rightarrow \langle w, Px \rangle$
  - Project documents to lower dimensional space
  - Easy to solve problem there (guarantees carry over)
- Nearest neighbor search
- Trace of matrix inverse $\operatorname{tr} PM^{-1}P^\top$

Locality Sensitive Hashing

- Basic idea
  Dimensionality reduction by multiplying with a dense random matrix (e.g. Gaussian, stable ...)

$$z = Px$$

- Document classification $\langle w, x \rangle \rightarrow \langle w, Px \rangle$
  - Project documents to lower dimensional space
  - Easy to solve problem there (guarantees carry over)
- Nearest neighbor search
- Trace of matrix inverse $\operatorname{tr} PM^{-1}P^\top$


Shingles

Shingles

- Basic idea (Broder et al. 1997)
  - Hash coordinates of nonzero vector
  - Keep k smallest hashes

The quick brown fox jumped over the lazy dog

http://cs.brown.edu/courses/csci2531/papers/nearduplicate.pdf

Shingles

- Basic idea (Broder et al. 1997)
  - Hash coordinates of nonzero vector
  - Keep k smallest hashes

The quick brown fox jumped over the lazy dog

9 23 11 3 10 42 9 14 99

http://cs.brown.edu/courses/csci2531/papers/nearduplicate.pdf

Shingles

- Basic idea (Broder et al. 1997)
  - Hash coordinates of nonzero vector
  - Keep k smallest hashes

The quick brown fox jumped over the lazy dog

9	23	11	3	10	42	9	14	99
The		brown	fox			the		

http://cs.brown.edu/courses/csci2531/papers/nearduplicate.pdf

Shingles

- Basic idea (Broder et al. 1997)
  - Hash coordinates of nonzero vector
  - Keep k smallest hashes

The quick brown fox jumped over the lazy dog

9 23 11 3 10 42 9 14 99

The brown fox the

The quick brown hedgehog had breakfast

http://cs.brown.edu/courses/csci2531/papers/nearduplicate.pdf

Shingles

- Basic idea (Broder et al. 1997)
  - Hash coordinates of nonzero vector
  - Keep k smallest hashes

The quick brown fox jumped over the lazy dog

9 23 11 3 10 42 9 14 99

The brown fox the

The quick brown hedgehog had breakfast

9 23 11 4 10 5

http://cs.brown.edu/courses/csci2531/papers/nearduplicate.pdf

Shingles

- Basic idea (Broder et al. 1997)
  - Hash coordinates of nonzero vector
  - Keep k smallest hashes

The quick brown fox jumped over the lazy dog

9 23 11 3 10 42 9 14 99

The brown fox the

The quick brown hedgehog had breakfast

9 23 11 4 10 5

The hedgehog breakfast

http://cs.brown.edu/courses/csci2531/papers/nearduplicate.pdf

Shingles

- Basic idea (Broder et al. 1997)
  - Hash coordinates of nonzero vector
  - Keep k smallest hashes

The	quick	brown	fox	jumped	over	the	lazy	dog
9	23	11	3	10	42	9	14	99
The		brown	fox			the		

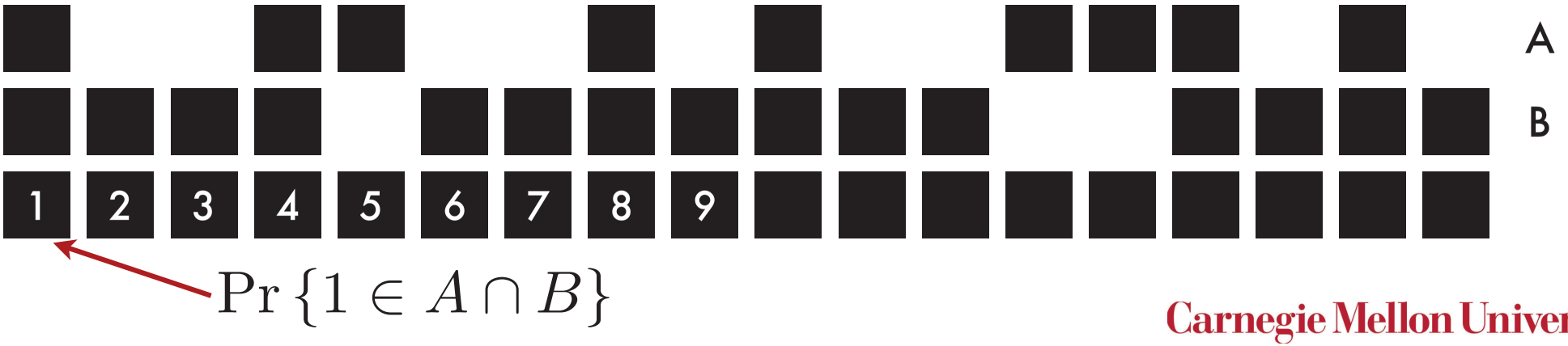
The	quick	brown	hedgehog	had	breakfast
9	23	11	4	10	5
The			hedgehog		breakfast

http://cs.brown.edu/courses/csci2531/papers/nearduplicate.pdf

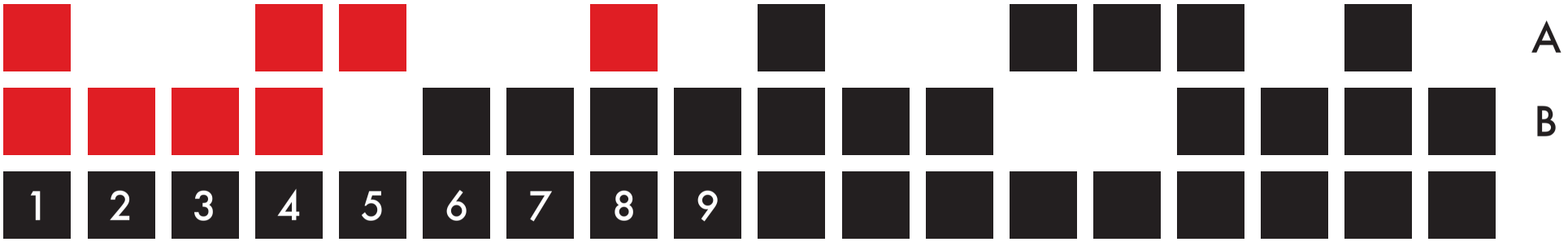
Shingles

- Basic idea (Broder et al. 1997)
  - Hash coordinates of nonzero vector
  - Keep k smallest hashes
- Estimate Jaccard coefficient by shingle overlap

$$r(A,B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$

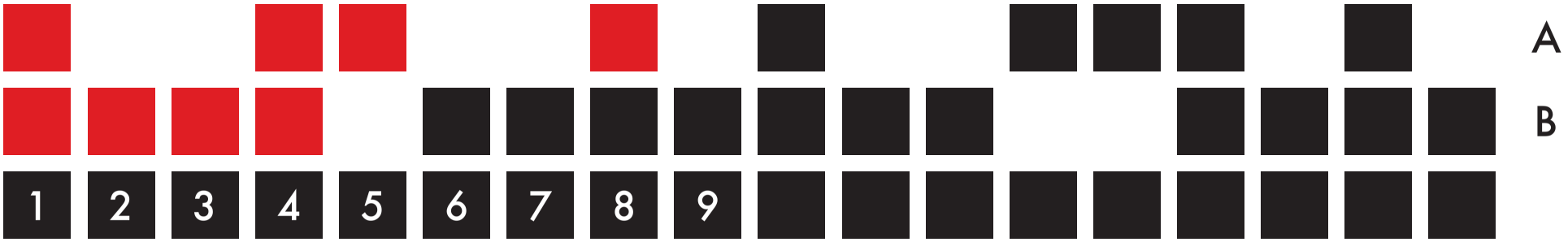


Conditional Random Sampling



- Basic idea (Li, Hastie, Church, 2006)
  http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2006_848.pdf
  - Value of hash lets us estimate nonzero density
  - Use this to estimate inner products (l_1, l_2, ...)
    - Smallest of the two largest hash values
    - Rescale inner product via $\frac{k}{\min(s(A), s(B))}$

Conditional Random Sampling

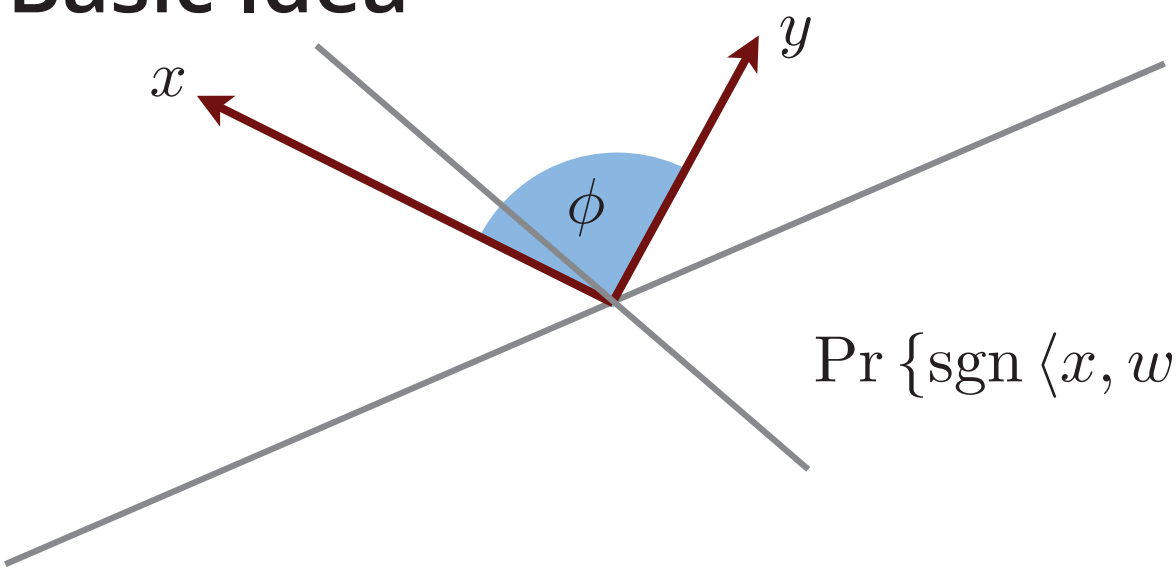


- Extreme sparsification
  - Only keep tiny fraction of vector around
  - Intelligent compression
    (exact for very sparse vector)
  - Much faster for many inner products
- Does not reduce dimensionality
- Aggregation over inner products nontrivial


Sim Hash

Sim Hash

- **Basic Idea**

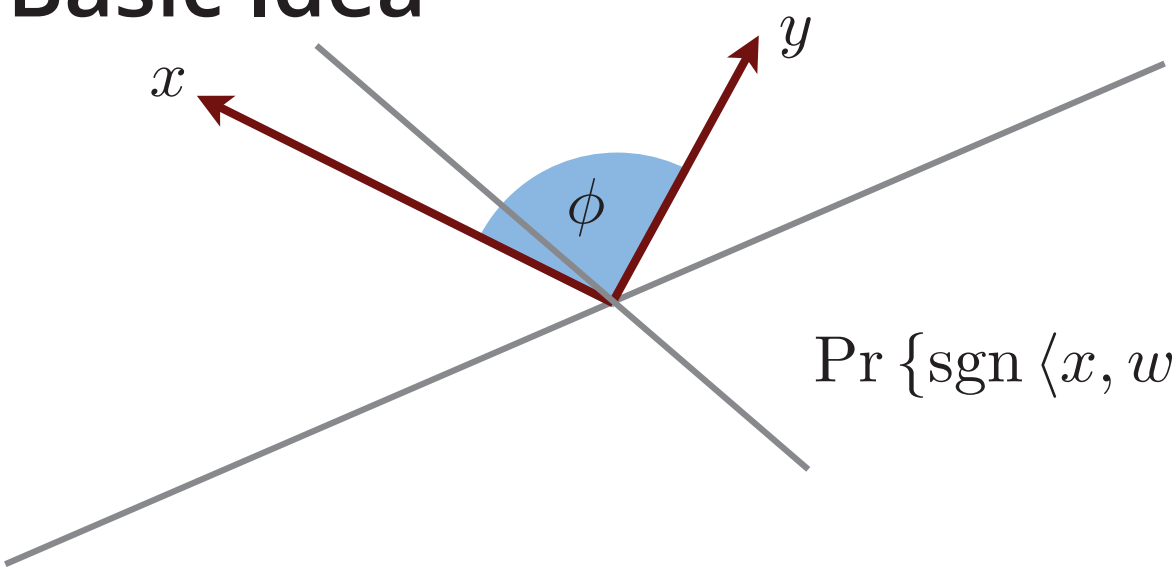


$$\Pr\{\operatorname{sgn}\langle x, w\rangle \neq \operatorname{sgn}\langle y, w\rangle\} = \frac{\phi}{\pi}$$

- **Goemans & Williamson, 1995**
  Use this for an SDP relaxation of graph cut
- **Charikar, 2003**
  Use this for hashing angles between vectors

Sim Hash

- **Basic Idea**



$$\Pr\{\operatorname{sgn}\langle x, w\rangle \neq \operatorname{sgn}\langle y, w\rangle\} = \frac{\phi}{\pi}$$

- **Hash map is very memory efficient (n bits)**

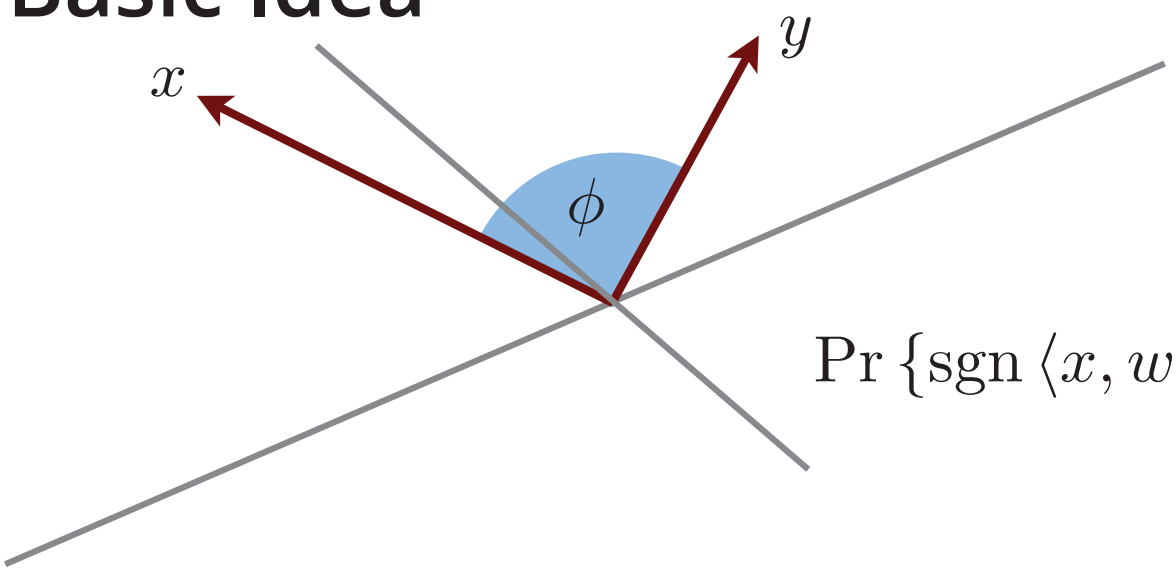
$$x \to h(x) = (\operatorname{sgn}\langle x, w_1\rangle, \ldots, \operatorname{sgn}\langle x, w_n\rangle)$$

- **Inner product estimation**

$$\langle x, y\rangle \approx \|x\| \|y\| \cos n^{-1} \|h(x) - h(y)\|_1$$

Sim Hash

- **Basic Idea**



$$\Pr\{\operatorname{sgn}\langle x, w\rangle \neq \operatorname{sgn}\langle y, w\rangle\} = \frac{\phi}{\pi}$$

- **Hash map is very memory efficient (n bits)**

$$x \to h(x) = (\operatorname{sgn}\langle x, w_1\rangle, \ldots, \operatorname{sgn}\langle x, w_n\rangle)$$

- **Inner product estimation**

$$\langle x, y\rangle \approx \|x\| \|y\| \cos n^{-1} \|h(x) - h(y)\|_1$$

CPUs are very fast

Clustering

- Gaussians are obviously a poor fit for text (e.g. negative word counts)
- Discrete distribution

$$p(x) = \exp\left(\langle w, x \rangle - g(w)\right)$$

normalization

exponential family

natural parameter

sufficient statistic

$$p(x|y) = \exp\left(\langle w_y, x \rangle - g(w_y)\right)$$

Clustering

- Gaussians are obviously a poor fit for text (e.g. negative word counts)
- Discrete distribution

$$p(x) = \exp\left(\langle w, x\rangle - g(w)\right)$$

normalization

exponential family

natural parameter

sufficient statistic

$$p(x|y) = \exp\left(\langle w_y, x\rangle - g(w_y)\right)$$

expensive

Clustering

- Gaussians are obviously a poor fit for text (e.g. negative word counts)
- Discrete distribution

$$p(x) = \exp\left(\langle w, x\rangle - g(w)\right)$$

- Sampling
  - Need to compute p(x|y) for many y
  - Big problem if we have >10,000 clusters

$$p(y_i|x_i) \propto p(y_i|y^{-i})p(x_i|y_i)$$

  - Approximate exponential family term

Clustering

- Sampler

$$s^l(x, y) = \|\theta_y\| \, \|\phi(x)\| \cos \pi z^l(\phi(x), \theta_y) - g(\theta_y) - \log n_y$$

  - Estimate inner product via sim hash
  - Reduce from O(d) to O(1)
  - Bits are much faster to handle than floats (this is why we can't use LSH)
- Upper bound (Chernoff) for rejection sampler (pretty bad acceptance probability)
- Better to use Metropolis Hastings proposal

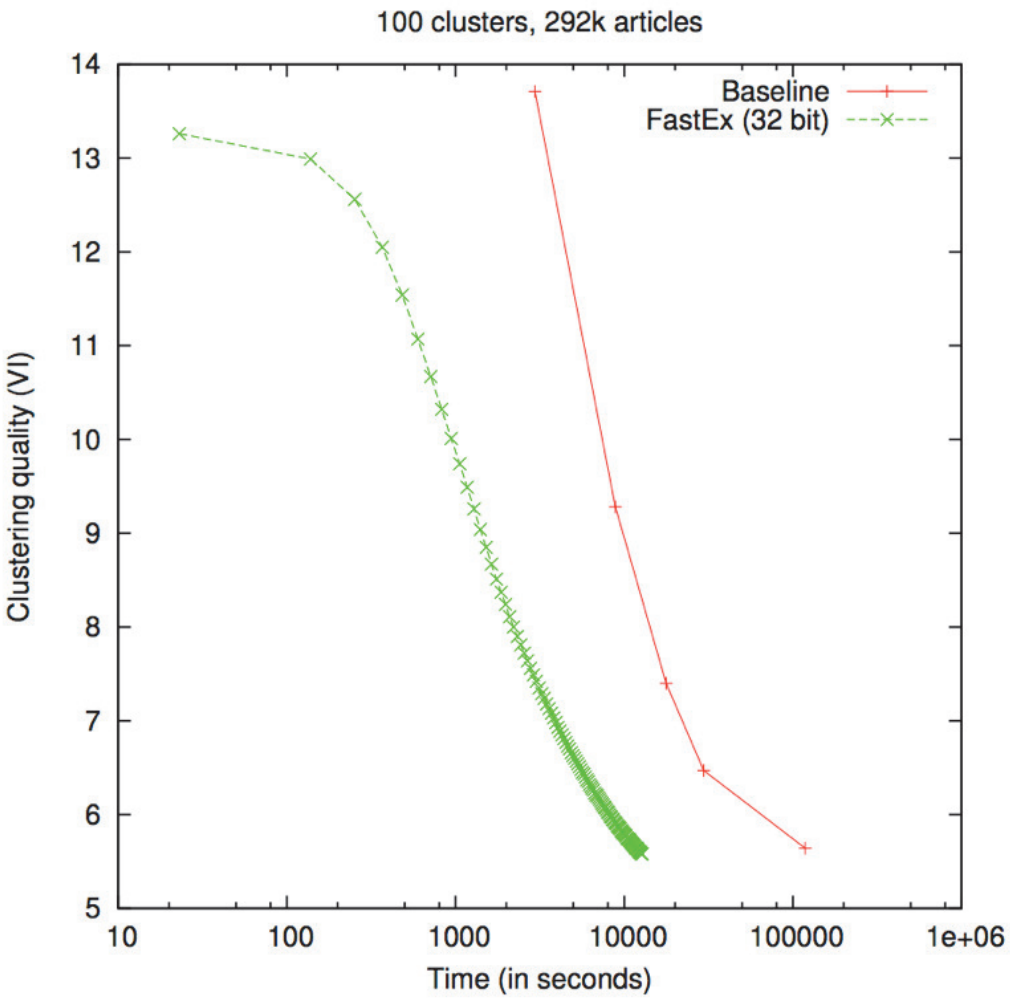
Clustering

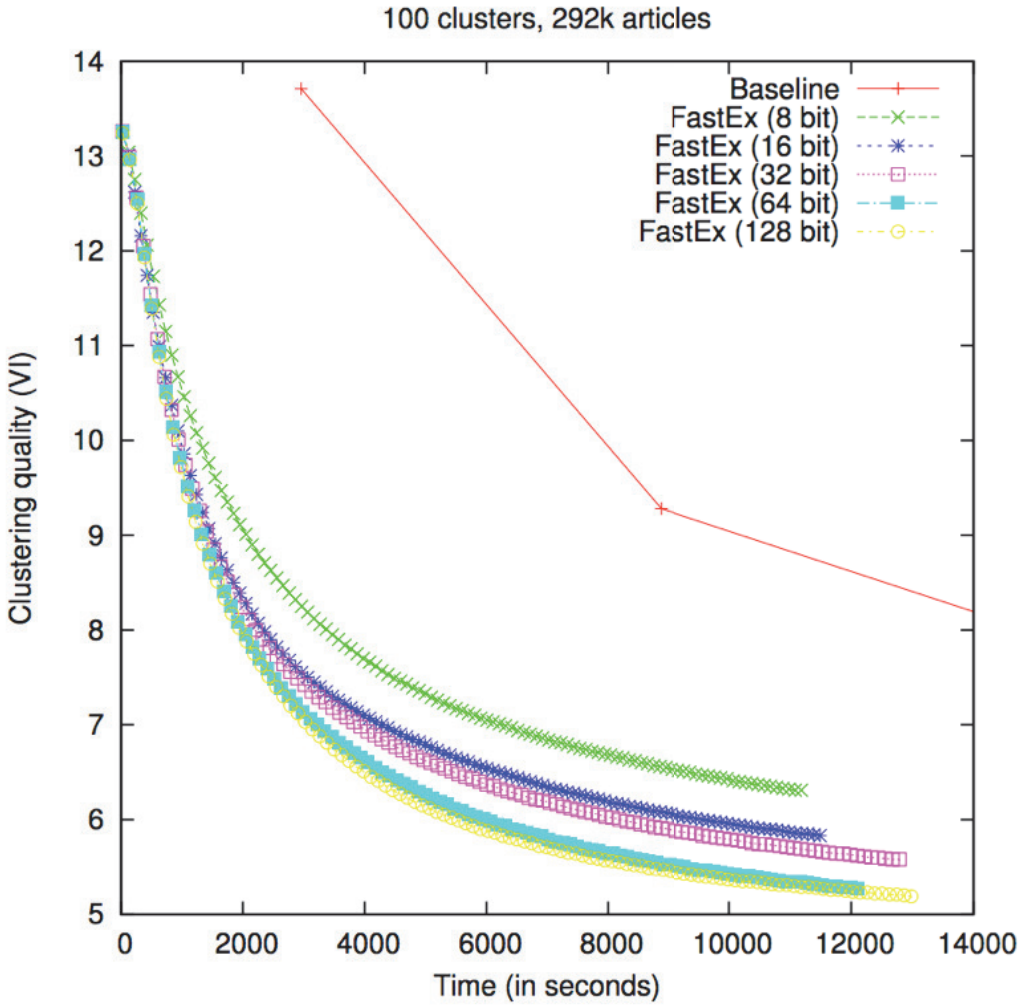
- Metropolis Hastings Sampler

$$q(y) \propto e^{\bar{s}^l(x,y)} \text{ and } r = \frac{q(y^{\text{old}})}{q(y_i^{\text{new}})} \frac{p(y_i^{\text{new}}) p(x_i | X^i_{y_i^{\text{new}}}, m_0, \mu_0)}{p(y_i^{\text{old}}) p(x_i | X^i_{y_i^{\text{old}}}, m_0, \mu_0)}$$

  - Cheap tricks to update hashes (cache floats)
  - Approximate conjugates via Taylor expansion
- AVX instructions Intel (256 bits/clock per core)
- Only need to compute exact inner product small number of times

Results





Ahmed et al., 2012 NIPS
http://books.nips.cc/papers/files/nips25/NIPS2012_1283.pdf

Results

- Time spent on proposal

	Clusters k	Bitsize l	**8**	**16**	**32**	**64**	**128**
Proposal	100		2.34	2.34	2.34	2.56	2.90
Total			69.52	69.52	78.77	81.16	82.19
Proposal	1000		18.80	18.80	18.80	21.42	29.12
Total			103.91	103.91	103.91	108.98	114.61

- Acceleration

Dataset	FastEx Quality (VI)	Baseline Quality (VI)	Speedup
W_{100}	5.04	5.60	9.25
W_{1000}	14.10	14.00	37.37

- Sparse Aggregators
  (Bloom, CountMin)
- Optimizing over Sparse Aggregators
  (Linear models, CoFi rank)
- Random function classes
  (Random Kitchen Sinks, Fast Food)
- Random Projections
  (LSH, SimHash, FastEx)